

**CoroWare CoroBot**  
**Wireless Connectivity**  
**(Draft #8)**



**Author:** Victor Fernandez

**Class:** CNT 4104 Software Project in Computer Networks

**Instructor:** Dr. Janusz Zalewski

FLORIDA GULF COAST UNIVERSITY

10501 FGCU Boulevard South

Fort Myers, FL 33965-6565

November 19, 2012

## **1. Introduction**

This is a new research project which involves CoroBot. CoroBot is a four wheeled robot created by Coroware with the goal of minimizing the complexity of robot development. CoroBot is equipped with a mini ITX motherboard and a PC-class CPU. One of the most important characteristic about CoroBot is that it has expansive program storage space and CPU capacity to run additional software. CoroBot was designed with developers in mind; all the hardware devices and CoroBot itself is supported by Microsoft Robotics Developer Studio. In addition, CoroBot's ample mounting space allows hardware developers to install additional hardware components such as GPS, laser range finder, environmental sensors and more. [1]

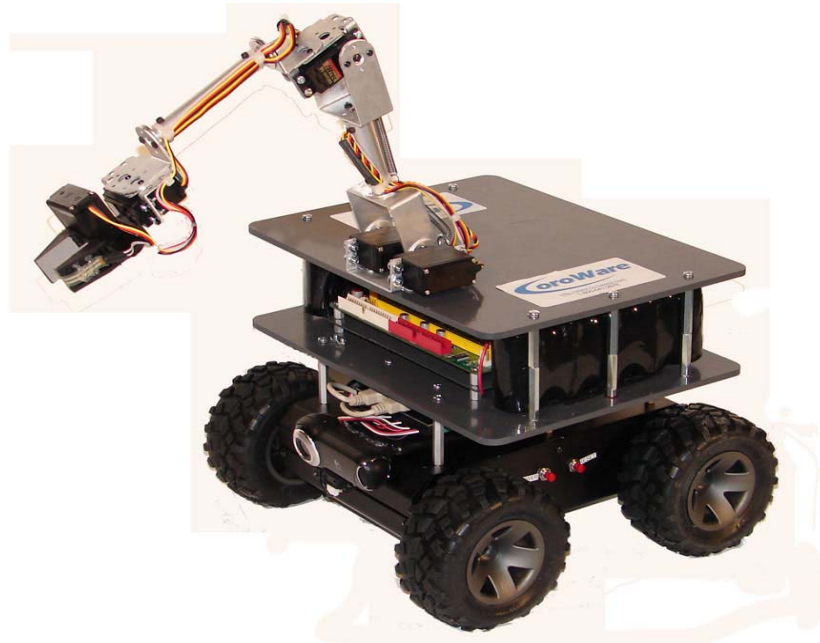
### **1.1 CoroBot's Hardware**

A Microsoft Kinect 3D sensor (Figure 1) is mounted in CoroBot's top deck.



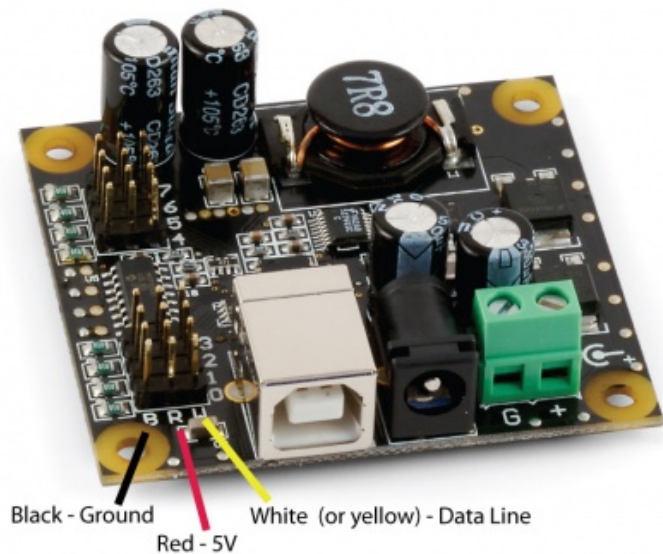
**Figure 1: Kinect. [4]**

A 4 DOF (Degrees of Freedom) robotic arm (Figure 2) is mounted in CoroBot's top deck.



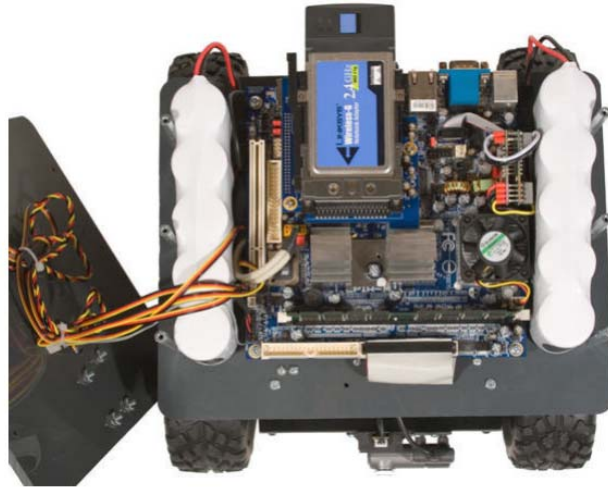
**Figure 2: CoroBot's robotic arm. [5]**

The robotic arm's servos are controlled by a Phidget 1061 Advanced Servo Controller 8-Motor (Figure 3).



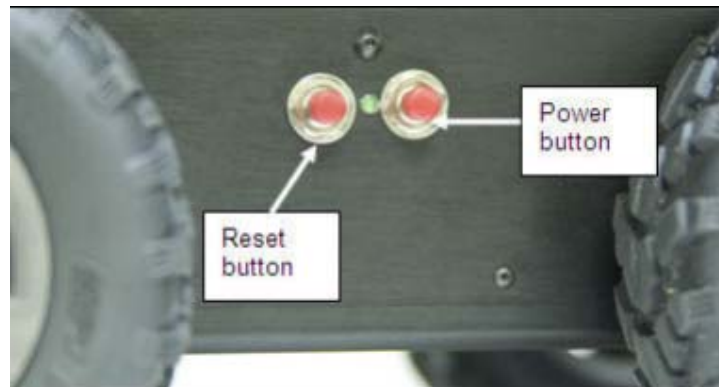
**Figure 3: Phidget 1061 Advanced Servo Controller 8-Motor. [6]**

CoroBot's upper deck (Figure 3) contains the mini ITX form factor motherboard and batteries. The current system in the lab is installed with dual boot configuration running Linux and Windows 7 operating system. The batteries' life span is approximately 2.5 hours and they take around six hours to fully charge. [2]



**Figure 4: Upper Deck. [2]**

The left side of CoroBot contains the Power button and Reset button. Each button is labeled accordingly (Figure 4).

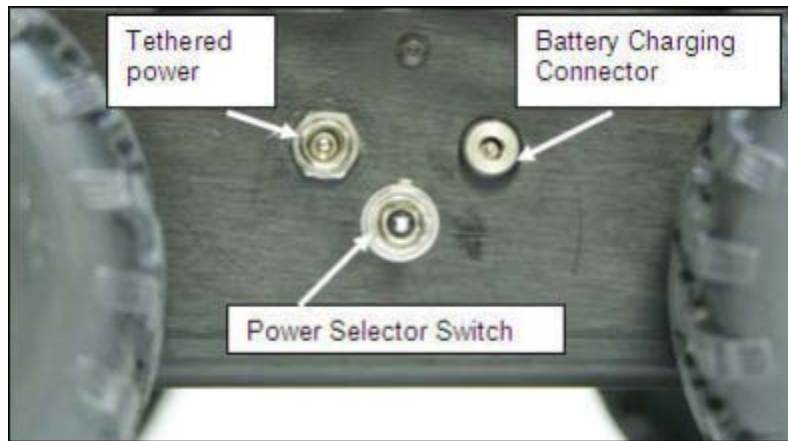


**Figure 5: Power and Reset buttons. [2]**

The right side of CoroBot contains the battery charging connector, tethered power connector, and power selector switch (Figure 5).

The power selector switch supports three positions. Each position is described as follow:

- Up: Operate in tethered power.
- Center: Off
- Down: Operate using battery power



**Figure 6: Right side of CoroBot. [2]**

To turn on CoroBot follow these steps:

1. Ensure the charger switch is in the 1.8 amps configuration.
2. Plug the battery charger to an AC outlet. Make sure the light flashes red/green, which indicates that the charger is receiving power.
3. Plug the power connector into the battery charging connector.
4. Change the power switch to the up position.
5. Press the power button (left side) and wait for the system to boot up.

## 1.2 Software Required

The following software is used in this project:

- Windows 7 operating system
- Microsoft Robotics Developer Studio 2008 R3

“Microsoft Robotics Developer Studio is a freely available .NET-based programming environment for building robotics applications. It can be used by both professional and non-professional developers as well as hobbyists.

In addition to providing support for Microsoft Visual Studio 2010, Microsoft Robotics Developer Studio 4 provides a Visual Programming Language (VPL) which allows developers to create applications simply by dragging and dropping components onto a canvas and wiring them together.” [3]

- Microsoft Visual Studio 2010

It is an integrated development environment (IDE) from Microsoft that assists developers in the creation of software.

- Microsoft .NET Framework
- Phidgets Drivers and Phidgets' C# API. [6]
- VLC media player

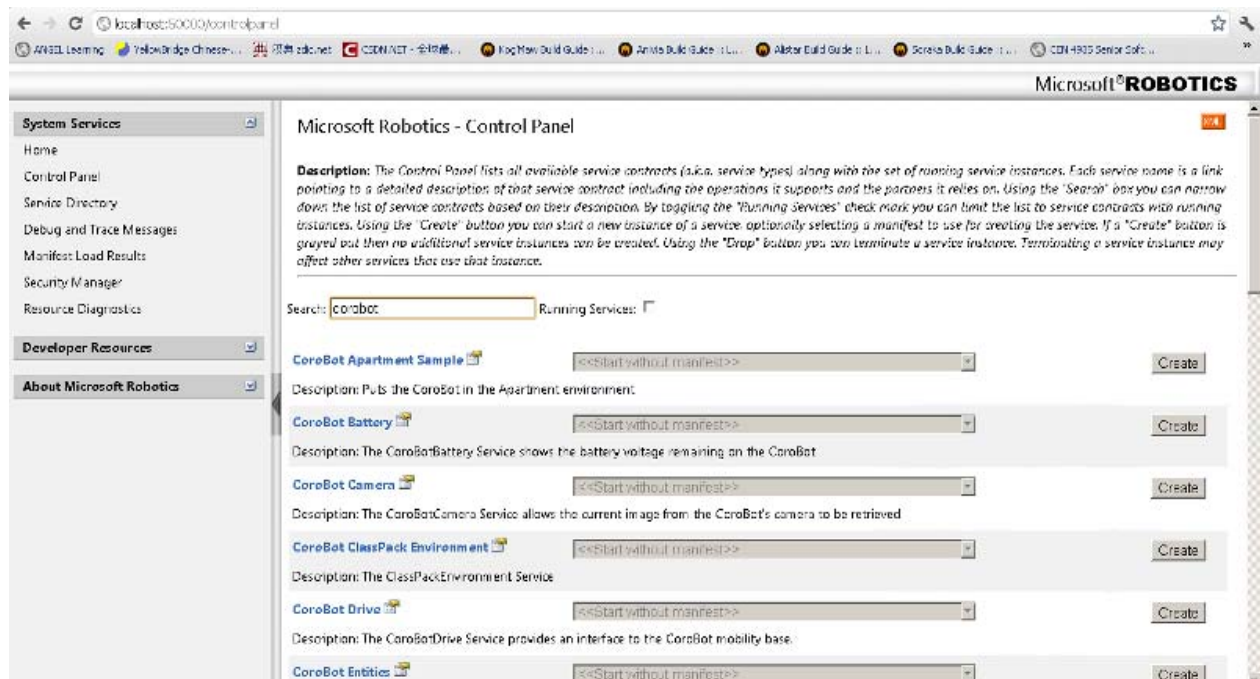
## 1.3 Previous Project

The previous team [7] installed and configured several programs that enabled them to remotely control CoroBot in Windows and Linux. To remotely control the CoroBot in Windows, the following software was installed:

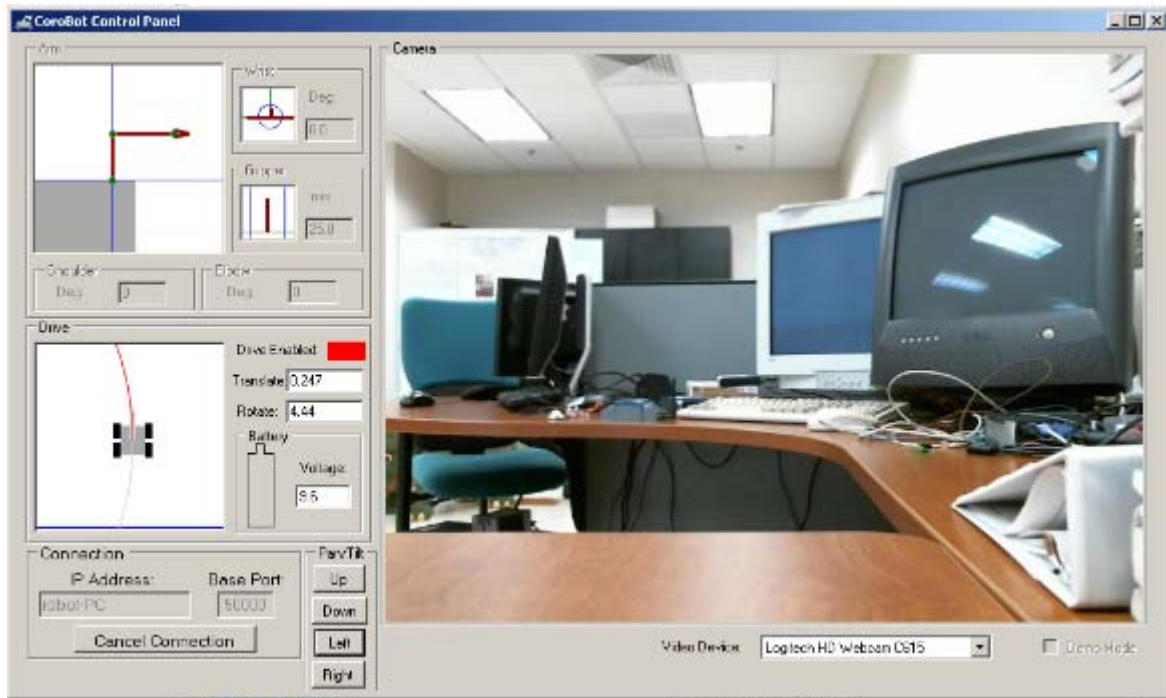
- Microsoft Windows XP or greater
- Microsoft .NET libraries 3.5
- Microsoft Visual Studio 2010 Professional
- Microsoft Robotics Developer Studio 2008 R3

- Phidgets drivers
- A Subversion client (Slik Subversion will do)

Once the required software was installed, the team downloaded and compiled CoroWare CoroBot Robotics Developer Studio Project. Then they used the DSS (Decentralized Software Services) host control panel (Figure 7) that comes with Microsoft Robotics Developer Studio to create several CoroBot services. Three of the CoroBot services can create a simulation environment in Microsoft Visual Simulation Environment to test CoroBot. Finally, to remotely control the CoroBot, the CoroBot OCU (Operator Control Unit) service was created. This service creates the CoroBot Control Panel (Figure 8), which enables the user to drive CoroBot from a remote location.



**Figure 7: The DSS Host Control Panel. [7]**



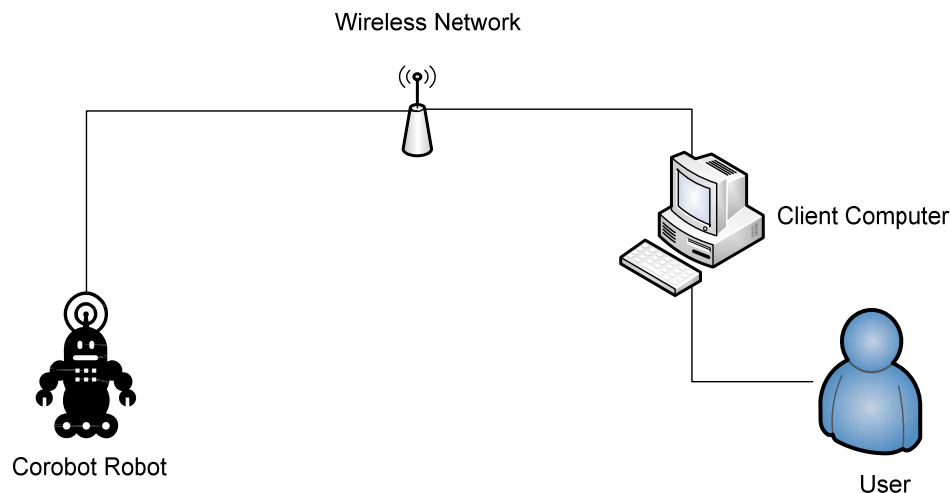
**Figure 8: CoroBot Control Panel used to remotely drive CoroBot. [7]**

## 2. Problem Definition

The objective of this project is to extend the work of the previous team by adding support to the new robotic arm mounted in CoroBot's top deck. The main goal of this project is to be able to remotely control the robotic arm, as well as see the video stream received from the robotic arm's web cam.

To accomplish this goal, a client and a server application has to be developed using Microsoft .NET framework. The client application shall allow the user to easily input the new position of the robotic arm. Through sockets, the client can connect to the server application and send the new position of the robotic arm. Once the server application received and processed the new position, it will move the robotic arm.

It is important to mention that the client application must be installed on the computer that will be remotely controlling the robotic arm. Likewise, the server application must be installed in CoroBot, which will act as the server. The diagram in Figure 9 illustrates the interaction between the user and CoroBot.



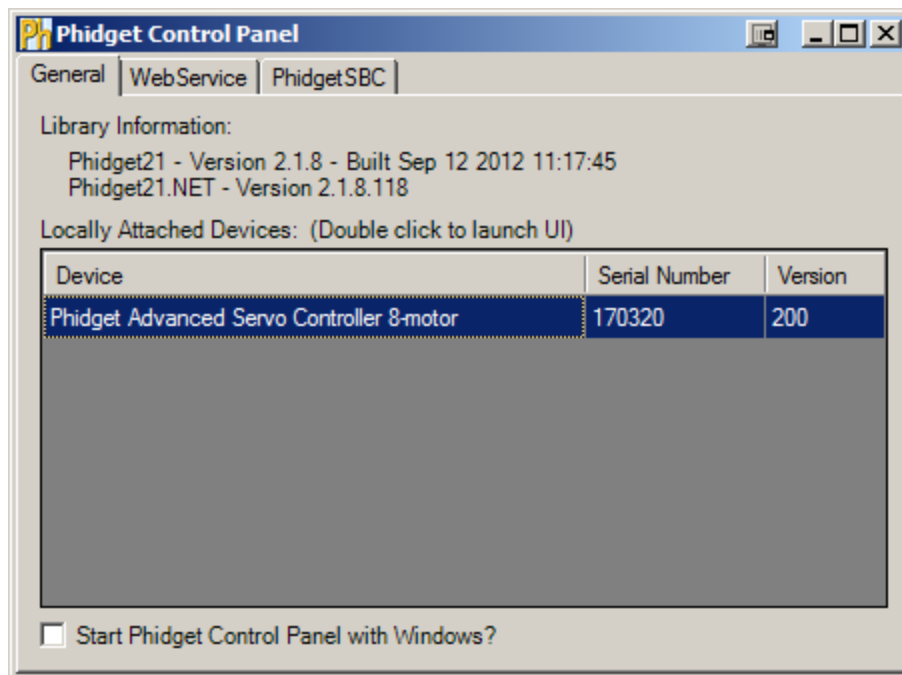
**Figure 9: Diagram demonstrating the interaction between the user and CoroBot.**

### 3. Prospective Solution

#### 3.1 Testing the Robotic Arm

Before starting to develop the client and server application, the robotic arm must be tested to see if it is working correctly. In order to test the robotic arm the following steps must be followed:

1. Install the Phidget drivers and libraries for Windows.
2. Once the Phidget libraries are installed, the Phidget icon will appear in the taskbar.  
Double click the icon to open the Phidget Control Panel (Figure 10).
3. Double click “Phidget Advanced Servo Controller 8-motor” under the listed devices to open the AdvancedServo-full example program (Figure 11). The example program can be used to test and calibrate the robotic arm.



**Figure 10: Phidget Control Panel**

**AdvancedServo-full**

AdvancedServo Control Details

Attached: ☒ True

Name: Phidget Advanced Servo Controller 8-motor

Serial No.: 170320

Version: 200

# Servos: 8

Servo Data

Choose Servo: 0

Type: HITEC\_HS322HD

Actual Velocity: 0

Actual Position: Unknown

Current: 0

☐ Engaged ☒ Speed Ramping ☐ Stopped

Set Target Position: 0

Set Velocity: 316

Set Acceleration: 182857.14

Set Max position: 180

Set Min Position: 0

Figure 11: AdvancedServo-full example program

## 3.2 Phidget API

After confirming that the robotic arm is functioning properly, it is time to start working with the robotic arm. As previously mentioned, the programming language that will be used to develop both the client and server application is C#. C# is capable of using the full Phidget API. The API provides several functions and events that enable programmers to control a Phidget. The following pages presents the API functions and events applicable for the Phidget 1061 Advanced Servo Controller 8-Motor as they appear in the Phidgets 1061 user guide. [6]

### Functions

- `int Count() [get]`  
“Returns the number of servos this PhidgetAdvancedServo can control. In the case of the 1061, this will always return 8. This call does not return the number of servos actually connected.”
- `double Acceleration(int ServoIndex) [get,set]`  
“Acceleration is the maximum change in velocity the PhidgetAdvancedServo uses when speeding up / slowing down a servo. The range of valid Acceleration is bounded by AccelerationMax/AccelerationMin. There is a practical limit on how fast your servo can accelerate, based on load and the physical design of the motor. This property should always be set by the user as part of initialization. The value does not initialize to the value last set on the device.”
- `double AccelerationMax(int ServoIndex) [get] : Constant`  
“AccelerationMax is the upper limit to which Acceleration can be set. For the 1061, this will always return 320000.”
- `double AccelerationMin(int ServoIndex) [get] : Constant`  
“AccelerationMin is the lower limit to which Acceleration can be set. For the 1061, this will always return 19.53125.”
- `double Velocity(int ServoIndex) [get]`

“Velocity returns the actual velocity that a particular servo is being driven at. A negative value means it is moving towards a lower position. This call does not return the actual physical velocity of the connected motor.”

- double VelocityLimit(int ServoIndex) [get, set]

“Gets or sets the maximum absolute velocity that the PhidgetAdvancedServo controller will drive the servo. If it’s changed mid-movement, the controller will accelerate accordingly. If the target position of the controller is near enough, then the VelocityLimit may never be reached. This property should always be set by the user as part of initialization. There is a practical limit on how fast your servo can rotate, based on the physical design of the motor. The range of VelocityLimit is bounded by VelocityMax/VelocityMin. Note that when VelocityLimit is set to 0, the servo will not move.”

- double VelocityMax(int ServoIndex) [get] : Constant

“VelocityMax is the absolute upper limit to which Velocity can be set. For the 1061, this will always return 6400.”

- double VelocityMin(int ServoIndex) [get] : Constant

“VelocityMin is the absolute lower limit to which Velocity can be set. For the 1061, this will always return 0.”

- double Position(int ServoIndex) [get,set]

“Position is used for both the target and actual position for a particular servo. If the servo is currently engaged and a new value is set, then the controller will continuously try to move to this position. Otherwise, this call will return the current position of the servo. This call does not return the actual physical position of the servo. The range of Position is bounded by PositionMin/PositionMax.

If the servo is not engaged, then the position cannot be read. The position can still be set while the servo is not engaged. Once engaged, the servo will snap to position if it is not there already.

This property should be set by the user as part of initialization. If not, it will report the last value set on the device (unless the 1061 has been power-cycled).

Get will return the last value as reported by the device. This means sets to this value will take a small amount of time to propagate.”

- `double PositionMax(int ServoIndex) [get,set]`

“PositionMax is the upper limit to which Position can be set, and is initialized to 233. It can be used to prevent the controller from going beyond a servo’s range of motion. A `PhidgetException` will be thrown if this is set above 233 or below PositionMin.”

- `double PositionMin(int ServoIndex) [get,set]`

“PositionMin is the lower limit to which Position can be set, and is initialized to -22.9921875. It can be used to prevent the controller from going beyond a servo’s range of motion. A `PhidgetException` will be thrown if this is set below -22.9921875 or above PositionMax.”

- `double Current(int ServoIndex) [get]`

“Current returns the power consumption in amps for a particular servo. The value returned for a disconnected or idle servo will be slightly above zero due to noise.”

- `bool SpeedRamping(int ServoIndex) [get,set]`

“SpeedRamping enables or disables whether the `PhidgetAdvancedServo` tries to smoothly control the motion of a particular servo. If enabled, then the 1061 will progressively send commands based on velocity, acceleration and position.

This property should be set by the user as part of initialization. If not, it will report the last value set on the device (unless the 1061 has been power-cycled).

Get will return the last value as reported by the device. This means sets to this value will take a small amount of time to propagate.”

- `bool Engaged(int ServoIndex) [get,set]`

“Enables a particular servo to be positioned. If this property is false, no power is applied to the motors. Note that when it is first enabled, the servo will snap to position, if it is not physically positioned at the same point. Engaged is useful for relaxing a servo once it’s reached a given position. If you are concerned about keeping accurate track of position, Engaged should not be disabled until Stopped = True.

This property should be set by the user as part of initialization. If not, it will report the last value set on the device (unless the 1061 has been power-cycled).

Get will return the last value as reported by the device. This means sets to this value will take a small amount of time to propagate.”

- `bool Stopped(int ServoIndex) [get]`

“Stopped returns false if the servo is currently in motion. It guarantees that the servo is not moving (unless you are moving it by hand), and that there are no commands in the pipeline to the servo. Note that virtually any API calls will cause Stopped to be temporarily false, even changing Acceleration or VelocityLimit on a stopped servo.”

- `Phidget_ServoType ServoType(int ServoIndex) [get,set]`

“Gets / Sets the servo type for an index. There is a list of some common servos that have been predefined by Phidgets Inc. This sets the PCM range (range of motion), the PCM to degrees ratios used internally and the maximum velocity. This allows the degree based functions to be accurate for a specific type of servo.

Note that servos are generally not very precise, so two servos of the same type may not behave exactly the same. Specific servo motors, as well as servos not in the list, can be independently quantified by the user and set up with the `setServoParameters` function.”

- `void setServoParameters(int ServoIndex, double MinUs, double MaxUs, double Degrees, double VelocityMax)`

“Sets the parameters for a custom servo motor. MinUs is the minimum PCM in microseconds, MaxUs is the maximum PCM in microseconds, Degrees is the degrees of rotation represented by the given PCM range and VelocityMax is the maximum velocity that the servo can maintain, in degrees/second.”

#### Events

- VelocityChange(int ServoIndex, double Velocity) [event]  
“An event issued when the velocity changes on a motor.”
- PositionChange(int ServoIndex, double Position) [event]  
“An event issued when the position changes on a motor.”
- CurrentChange(int ServoIndex, double Current) [event]  
“An event issued when the current consumed changes on a servo.”

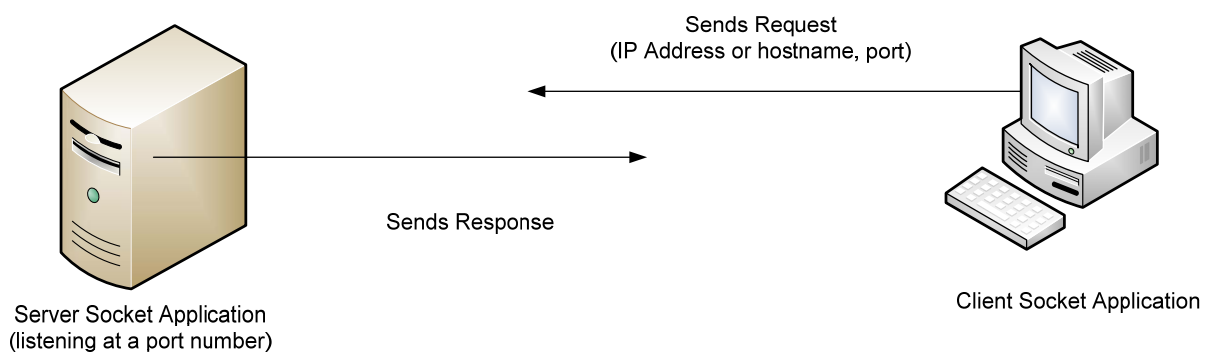
### 3.3 Socket Communication

In addition to being able to completely use the Phidgets API, C# also provides a managed implementation of sockets through the System.Net.Sockets Namespace. The Socket class allows developers to perform both synchronous and asynchronous network communication using several protocols such as IP, TCP, and UDP. [8]

A socket is an endpoint of a bidirectional communication between two applications. One of the applications is the C# client socket, and the other application is the C# server socket. In essence, describing socket communication is very simple.

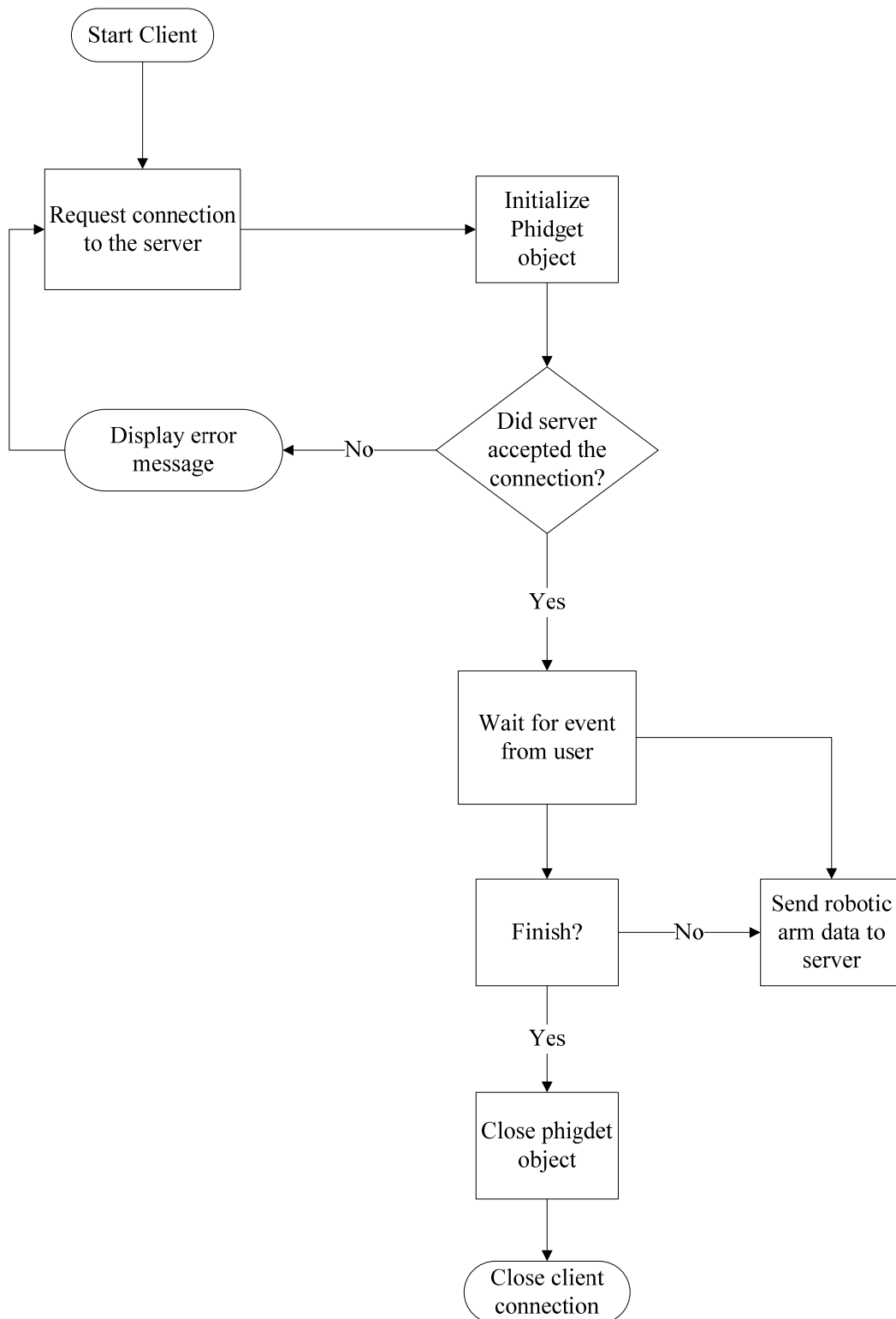
The server socket application binds to a port number on the computer, in this case CoroBot, and listens for incoming requests from a client socket application. In order for the client socket application to connect to the server socket, it must know the IP (Internet Protocol) address or the hostname of the computer where the server socket application is running, as well as the port number where the server socket is listening.

Once the server application accepted and established connection with the client application, they can proceed to exchange information. Figure 12 illustrates the communication via sockets between a client and server application.



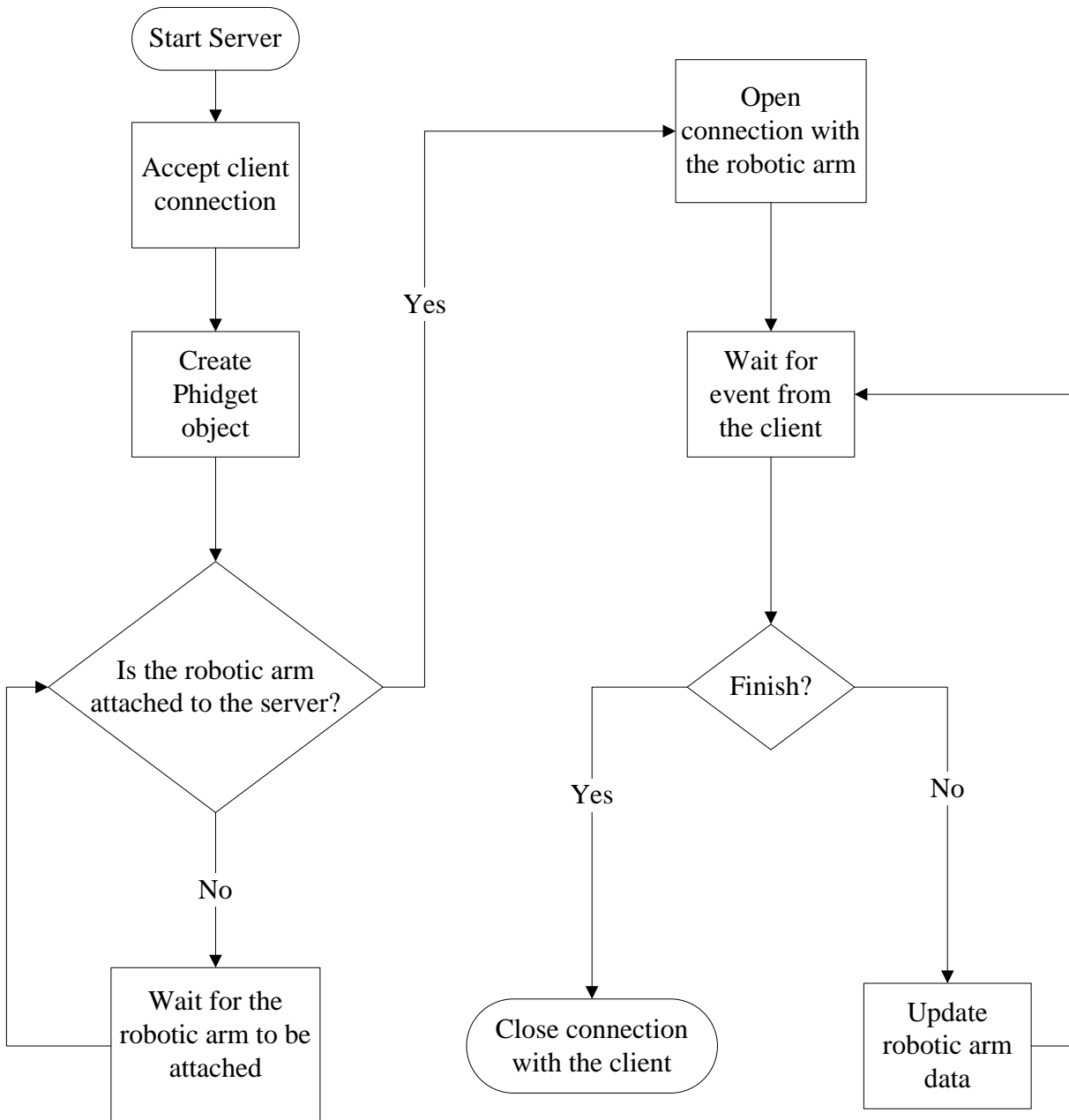
**Figure 12: Socket Communication**

Figure 13 shows a detailed flowchart of the client operations.



**Figure 13: Client operations**

Figure 14 shows a detailed flowchart of the server operations.



**Figure 14: Server operations**

## 4. Implementation

The client application has a GUI (Graphical User Interface) that allows the user to remotely change the position and other attributes for each servo of the robotic arm. To accomplish this, the user must first enter the following information in the “Server Configuration” tab of the client (Figure 15):

1. Server IP address

The IP address of the remote computer to which the robotic arm is physically connected (CoroBot).

2. Port Number

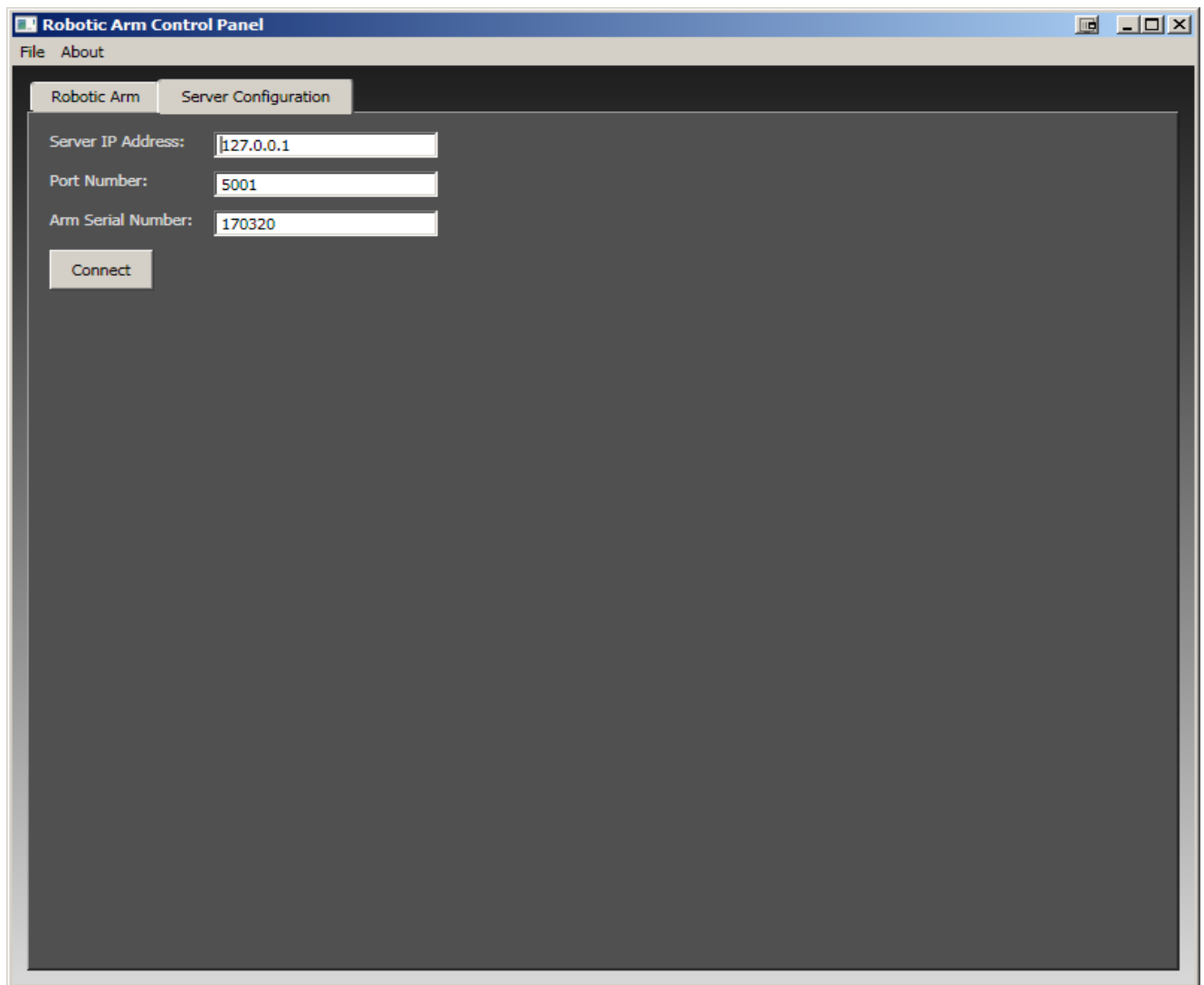
The port number used during the communication between the client and the server.

3. Serial Number

The serial number of the robotic arm. This number is required before establishing the connection with the server because there could be several robotic arms connected to the server, and the client would not know which robotic arm the user wants to control.

Upon entering this information, the user may click the “Connect” button. The client will now try to establish connection to the previously specified server and open a connection with the robotic arm. Once the client successfully connected to the server, the user will be able to change the position of the robotic arm. Because the client is working behind a GUI, the code is event driven. In other words, the client responds to user actions such as, mouse clicks, key presses, and drags of sliders.

The “Robotic Arm” tab (Figure 16) in the client is where the main interaction between the user and the client application occurs. The “Robotic Arm” tab functionality is described in the following pages.



**Figure 15: Client's Server Configuration Tab**

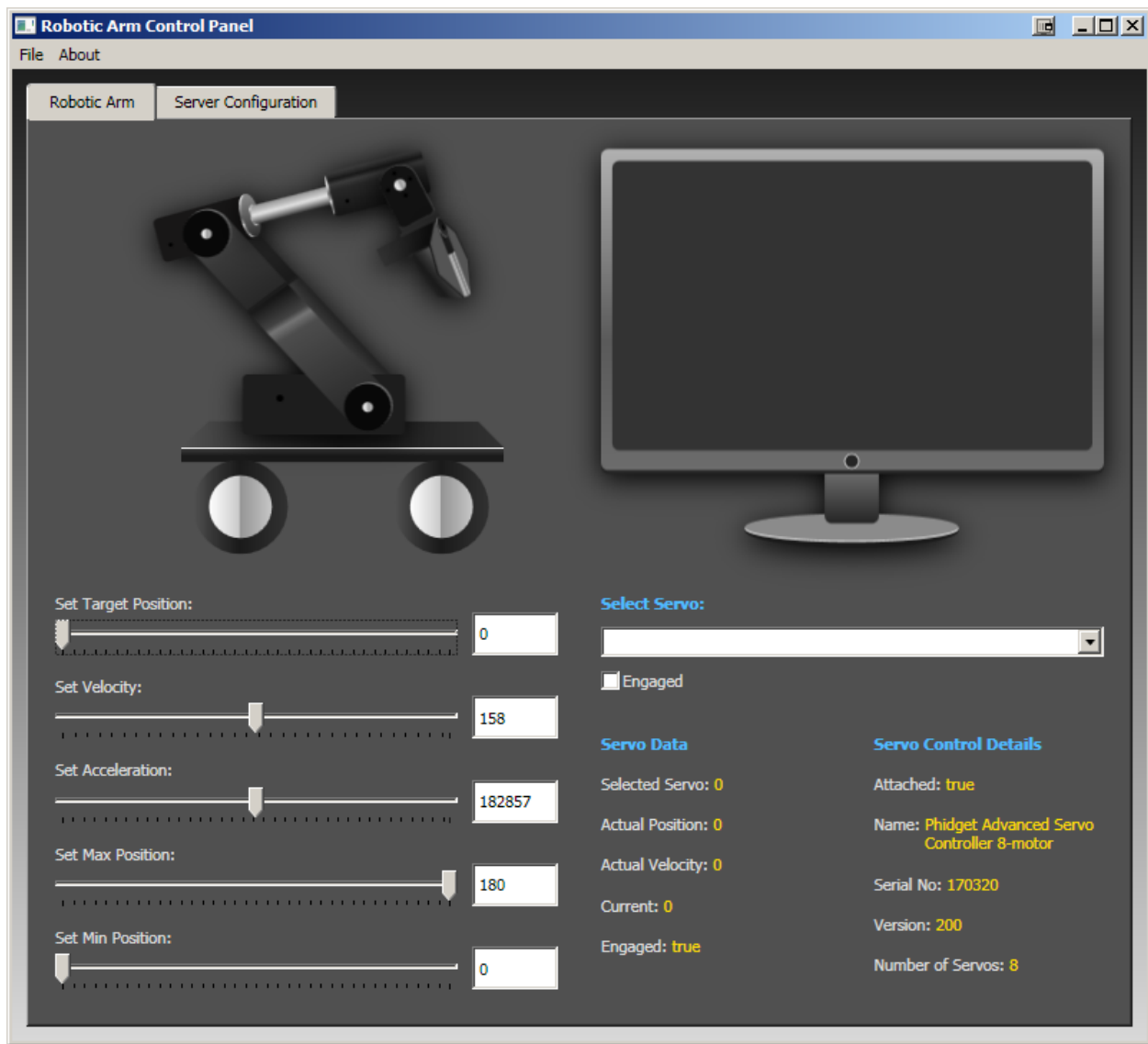


Figure 16: Robotic Arm Tab

The Robotic Arm tab GUI is divided into four quadrants, which are essentially two columns and two rows (figure 17).

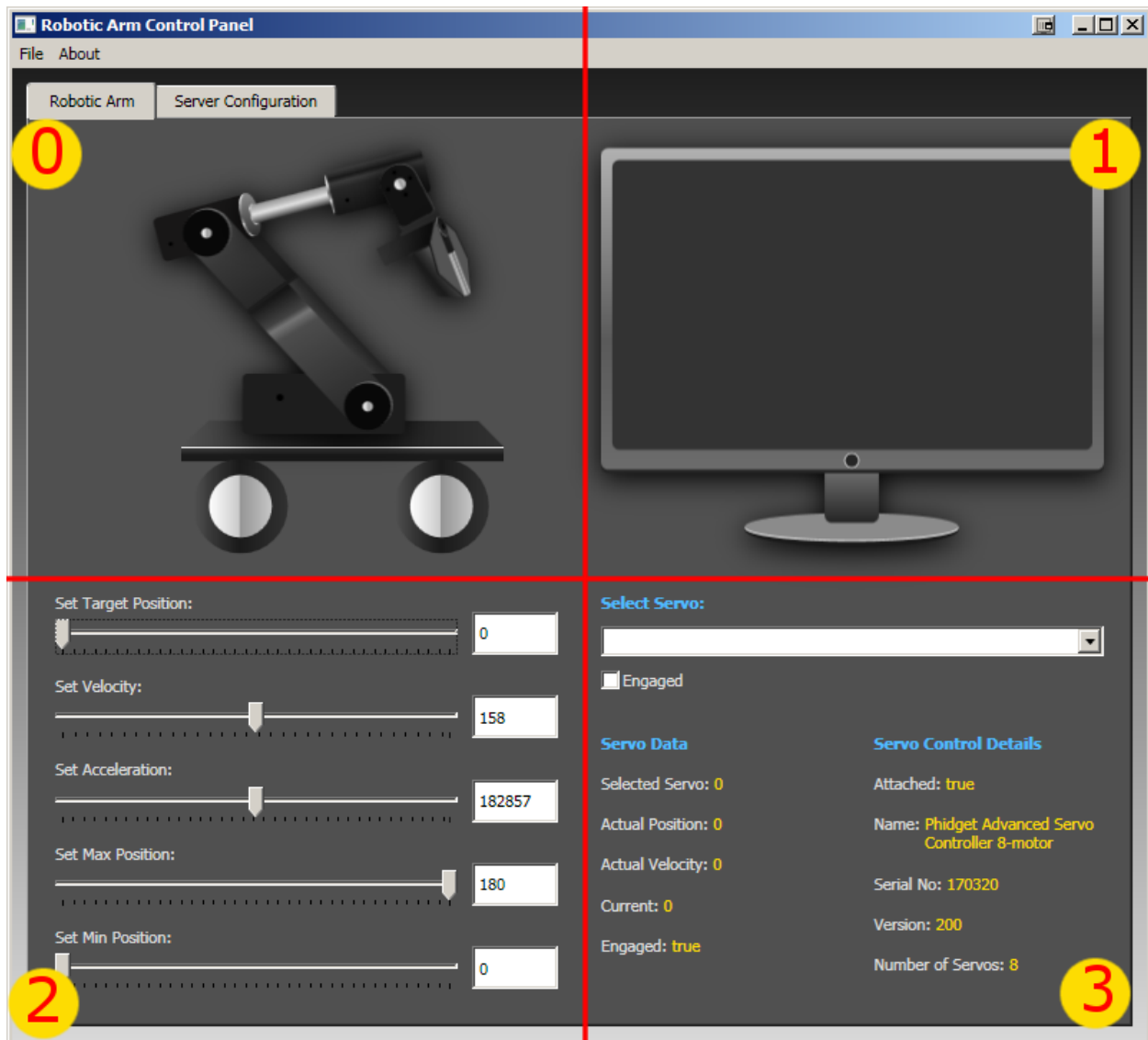
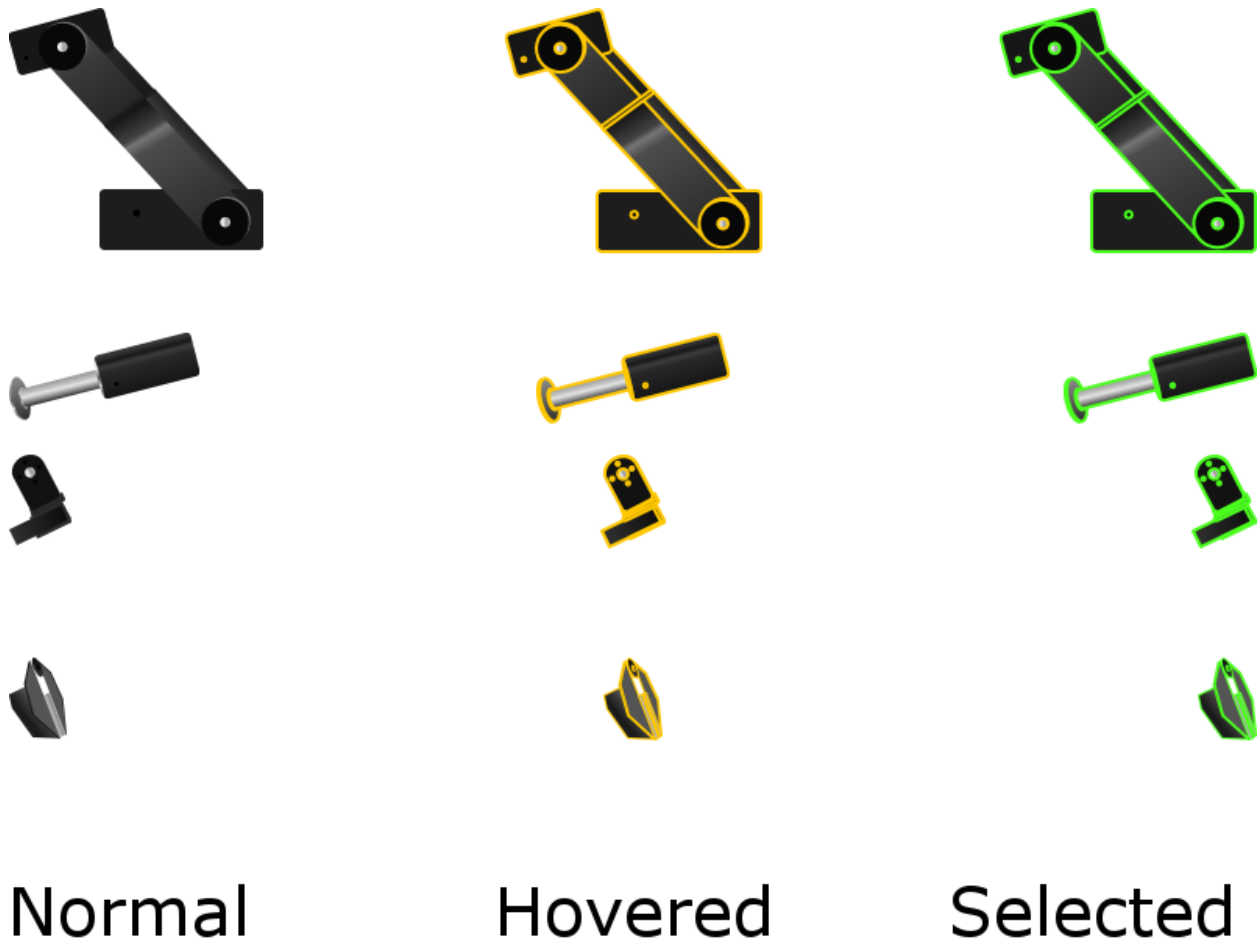


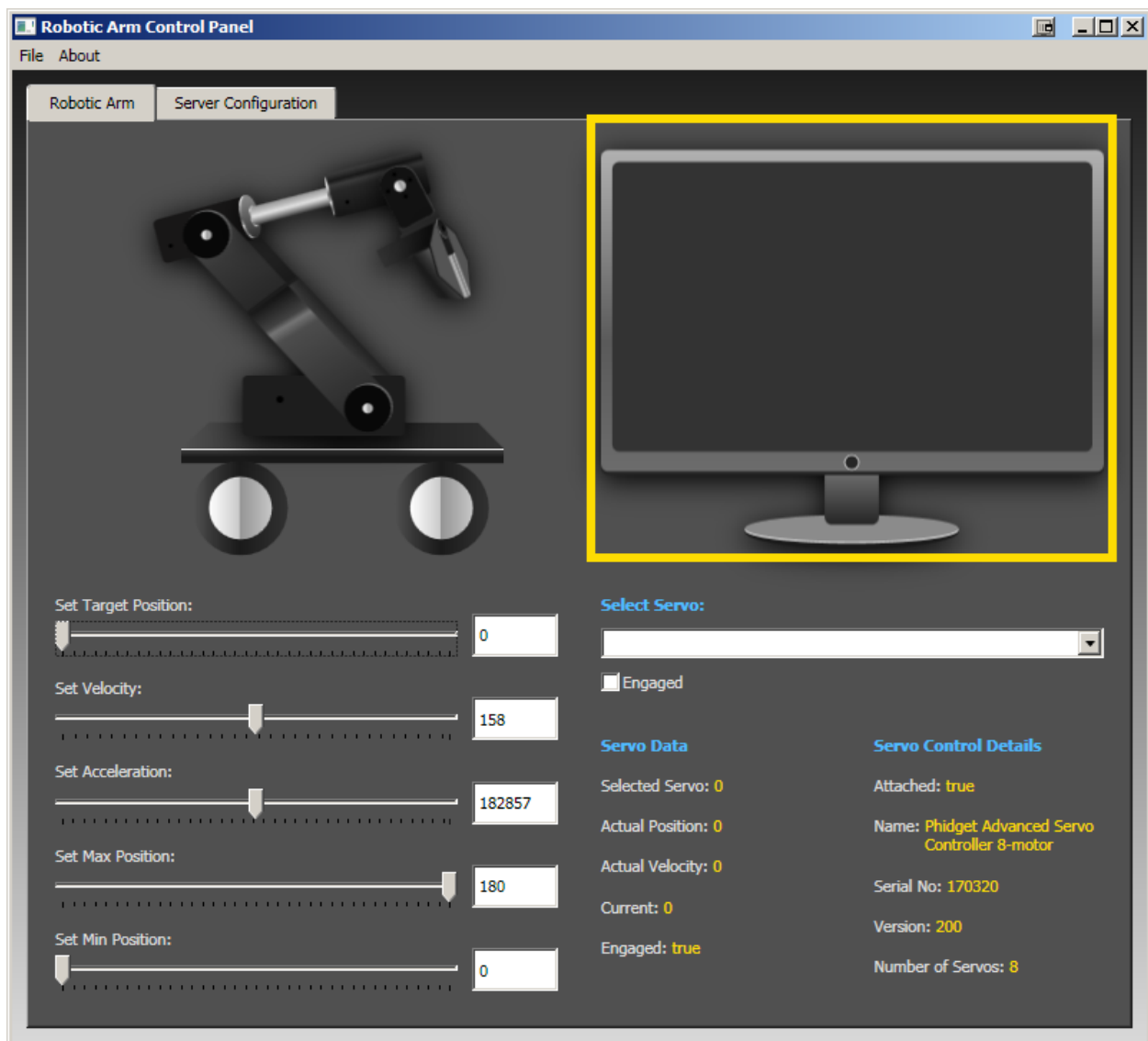
Figure 17: Robotic Arm Tab Layout

In the first quadrant (quadrant 0) a vector graphic representation of the robotic arm, which is mounted in CoroBot's top deck, is displayed. Each servo in the vector graphic is independent of each other. This allows the user to easily select a servo by just placing the mouse pointer over it and left-clicking. Each servo has three states: normal, mouse hovered, and selected (Figure 18).



**Figure 18: Servo States. Vector art designed by Victor Fernandez.**

In the second quadrant (quadrant 1), a vector graphic representation of a monitor is displayed (Figure 19). The video stream from the camera that is mounted in the robotic arm will appear inside the monitor. The MediaElement object, which is part of the .NET Framework, is used to capture the video stream. But in order to capture the video stream from the camera, the camera needs to be broadcasted over the network first. A step by step guide on how to broadcast the robotic arm's web camera appears in Appendix B-Broadcast Web Camera with VLC Media Player.



**Figure 19: Monitor Vector**

In the third quadrant (quadrant 2) five slider controls are present (Figure 20). Each slider allows the user to change a property of the selected servo. When the client detects that the user changed any of the sliders, the corresponding Phidget API method or event is invoked.

Sliders:

- 1) Set Target Position: changes the actual position (degrees) of a servo. By default, the minimum value supported by this slider is 0, and the maximum value is 180. However, the user may change the bounding values of this slider by dragging the Set Max Position or the Set Min Position slider.
- 2) Set Velocity: changes the velocity of a servo. If the velocity is 0, the servo will not move. The minimum value supported by this slider is 0, and the maximum value is 316.
- 3) Set Acceleration: changes the acceleration of a servo. The minimum value supported by this slider is 22, and the maximum value is 320,000.
- 4) Set Max Position: changes the maximum value of the Set Target Position slider. The minimum value supported by this slider is 0, and the maximum value is 180.
- 5) Set Min Position: changes the minimum value of the Set Target Position slider. The minimum value supported by this slider is 0, and the maximum value is 180.

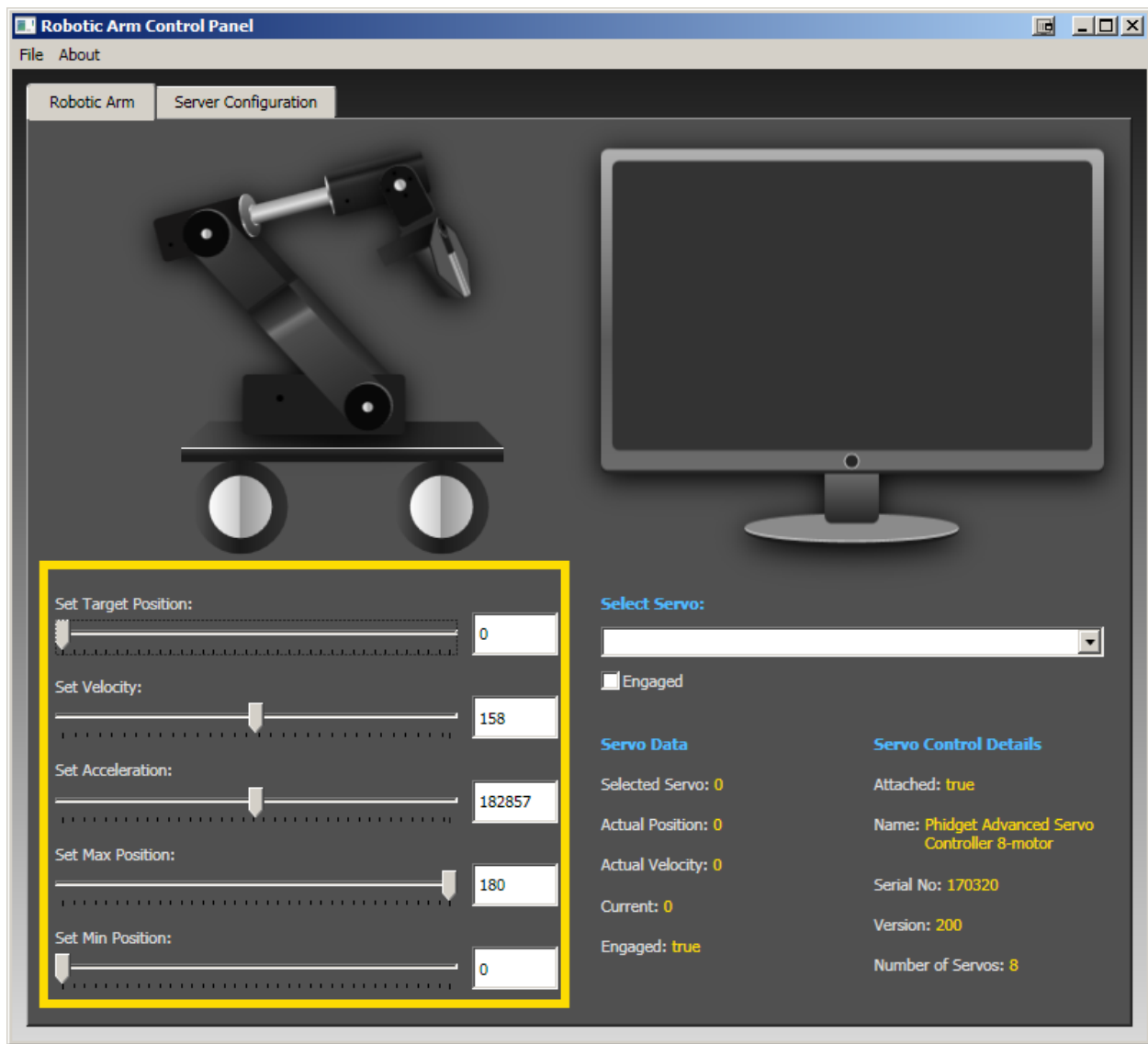


Figure 20: Servo Property Sliders

In the fourth quadrant (Figure 21), several controls are present. The first control is a combo box which allows the user to select a servo of the robotic arm. This combo box provides the same functionality as the robotic arm vector graphic located in first quadrant.

The second control, located below the combo box, is a check box that allows the user to change the engaged property of a servo. If a servo is not engaged, no power is applied to it, thus the servo will not react to the changes of the Set Target Position slider. When the client application successfully established a connection with the server and a robotic arm is attached, the engaged property for all the servos will be set to true.

Additional data about each servo and the servo controller is displayed below the Engaged check box.

The Servo Data group displays the following information about the currently selected servo:

- 1) its index number
- 2) Its actual position
- 3) Its actual velocity
- 4) The electrical current flowing through the servo
- 5) The value of the engaged property (true or false)

The Servo Control Details group displays the following information about the Phidget controller:

- 1) The value of the attached property (true or false)
- 2) Its name
- 3) Its serial number
- 4) Its version number
- 5) The maximum number of servos supported by the controller. The 1061 controller always return 8.

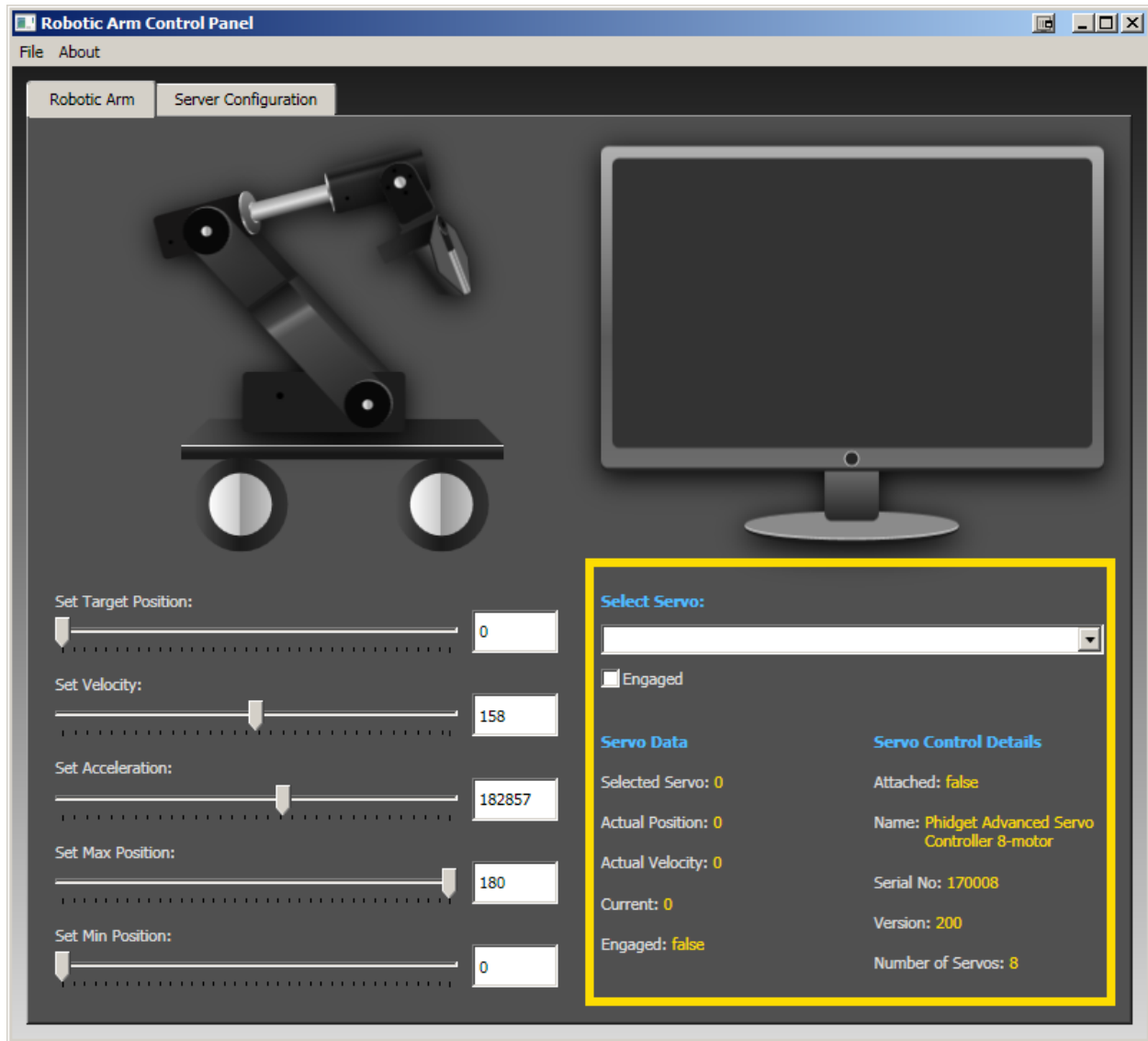


Figure 21: Fourth Quadrant

## **Conclusion**

By using the Phidget API and the .NET Framework, the Robotic Arm Control Panel client application is able to remotely control the robotic arm that is mounted in CoroBot's top deck. The graphical user interface displays a vector graphic representation of the robotic arm which allows the user to easily select a servo of the robotic arm. Once a servo is selected, the user may change several properties of the servo, such as its position. In addition, the Robotic Arm Control Panel displays the video stream from the robotic arm's web cam that is produced by VLC media player. Unfortunately, there is a delay of 3 to 5 seconds in the display.

This project could be expanded by adding the ability to move the robotic arm with a wireless joystick or controller.

The main difficulty that I faced during the development of this project was that I did not have any documentation or previous experience in how to program a robot. Fortunately, Phidget provides a really well documented C# API which helped me start programming the robotic arm.

## **Bibliography**

- [1] R. D. Inc. URL: <http://www.robotshop.com/PDF/coroware-corobot-pamphlet.pdf>.
- [2] I. CoroWare, Corobot User Guide, URL: <http://www.robotshop.com/PDF/corobot-user-guide.pdf>.
- [3] Microsoft, "Microsoft Robotics Developer Studio 4," 6 March 2012, URL: <http://www.microsoft.com/en-us/download/details.aspx?id=29081>.
- [4] PCWorld, "Software-Microsoft Kinect," URL: <http://www.pcworld.com/product/714802/kinect.html>.
- [5] T. R. MarketPlace, "CB-COROBOT," URL: <http://www.robotmarketplace.com/products/CB-COROBOT.html>.
- [6] P. Inc., "1061 User Guide," 7 September 2012. URL: [http://www.phidgets.com/wiki/images/thumb/6/6b/1061\\_0\\_Pins.jpg/400px-1061\\_0\\_Pins.jpg](http://www.phidgets.com/wiki/images/thumb/6/6b/1061_0_Pins.jpg/400px-1061_0_Pins.jpg).
- [7] X. Zhu, A. Piñeiro, "CoroWare CoroBot," Senior Software Engineering Project, FGCU, Fort Myers, 2012.
- [8] Microsoft, "Socket Class," URL: [http://msdn.microsoft.com/en-us/library/system.net.sockets.socket\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/system.net.sockets.socket(v=vs.100).aspx).

## Appendix A. Source Code

Window1 class:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

//Needed for the AdvancedServo class, phidget base classes, and PhidgetException class
using Phidgets;
//Needed for the event handling classes
using Phidgets.Events;
using System.Net;

namespace Robotic_Arm_Control_Panel
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>
    public partial class Window1 : Window
    {
        AdvancedServo advServo;
        Servo[] selectedServoArray;
        int NUMBER_OF_SERVOS = 5; //The actual number of servos installed
        int selectedServoIndex = 0;
        AboutWindow aboutWindow;

        public Window1()
        {

            //Advanced Servo object
            advServo = new AdvancedServo();

            //Array of 5 servo objects. Each element represents
```

```

//a robotic arm servo
selectedServoArray = new Servo[NUMBER_OF_SERVOS];

//instantiate the 5 servos of the robotic arm
for (int i = 0; i < NUMBER_OF_SERVOS; i++)
{
    selectedServoArray[i] = new Servo();
}

//Hook the basic event handlers
advServo.Attach += new Phidgets.Events.AttachEventHandler(advServo_Attach);
advServo.Detach += new Phidgets.Events.DetachEventHandler(advServo_Detach);
advServo.Error += new Phidgets.Events.ErrorEventHandler(advServo_Error);

//hook the phidget specific event handlers
advServo.PositionChange += new Phidgets.Events.PositionChangeEventHandler
    (advServo_PositionChange);
advServo.VelocityChange += new Phidgets.Events.VelocityChangeEventHandler
    (advServo_VelocityChange);

InitializeComponent();
}

//Attach event handler. Display serial number of the attached servo device
void advServo_Attach(object sender, AttachEventArgs e)
{
    Console.WriteLine("AdvancedServo {0} attached!",
        e.Device.SerialNumber.ToString());
    AdvancedServo attached = (AdvancedServo)sender;

    //Set the default servo type to the one Phidgets sells
    foreach (AdvancedServoServo motor in attached.servos)
    {
        motor.Type = ServoServo.ServoType.HITEC_HS322HD;
        motor.SpeedRamping = true;
    }

    //configure the initial position of each servo
    attached.servos[0].Position = 145;
    selectedServoArray[0].Position = 145;

    attached.servos[1].Position = 110;
    selectedServoArray[1].Position = 110;

    attached.servos[2].Position = 180;
    selectedServoArray[2].Position = 180;
}

```

```

    attached.servos[3].Position = 70;
    selectedServoArray[3].Position = 70;

    attached.servos[4].Position = 83;
    selectedServoArray[4].Position = 83;

    //engage the servos. each servo will move to
    //its initial position
    for (int i = 0; i < NUMBER_OF_SERVOS; i++)
    {
        selectedServoArray[i].Engaged = true;
        attached.servos[i].Engaged = true;
    }
}

//Detach event handler. Display the serial number of the detached servo device
static void advServo_Detach(object sender, DetachEventArgs e)
{
    Console.WriteLine("AdvancedServo {0} detached!",
        e.Device.SerialNumber.ToString());
}

//Error event handler. Display the error description to the console
static void advServo_Error(object sender, ErrorEventArgs e)
{
    Console.WriteLine("AdvancedServo Error: {0}", e.Description);
}

//Position Change event handler. Display which servo changed position and its
//new position value to the console
static void advServo_PositionChange(object sender, PositionChangeEventArgs e)
{
    Console.WriteLine("Motor #{0} - Actual Position: {1}",
        e.Index, e.Position.ToString());
}

//Velocity Change event handler. Display which servo's velocity is changing and
//its new value to the console
static void advServo_VelocityChange(object sender, VelocityChangeEventArgs e)
{
    Console.WriteLine("Motor #{0} - Actual Velocity: {1}", e.Index,
        e.Velocity.ToString());
}

#region Window1 GUI Event Handlers

```

```

private void ProgramWindow_Loaded(object sender, RoutedEventArgs e)
{
    if (advServo != null)
    {
    }
}

// Invoked when the "Connect" button is clicked
private void OpenButton_Click(object sender, RoutedEventArgs e)
{
    if (advServo != null)
    {

        try
        {
            IPAddress serverAddress = IPAddress.Parse(ServerIpTextBox.Text);
            int port = int.Parse(PortNumberTextBox.Text);
            int serialNumber = int.Parse(SerialNumberTextBox.Text);

            advServo.open(serialNumber, serverAddress.ToString(), port);
        }
        catch (Exception)
        {

        }
    }
}

// Invoked when the position slider is dragged
private void PositionSlider_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<double> e)
{
    //Move the selected servo to the specified position
    if (advServo.Attached)
    {
        selectedServoArray[selectedServoIndex].Position =
Math.Truncate(PositionSlider.Value * 100) / 100;
        advServo.servos[selectedServoIndex].Position =
selectedServoArray[selectedServoIndex].Position;
    }
}

// Invoked when the user changes servos

```

```

private void ServoComboBox_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    selectedServoIndex = ((sender as ComboBox).SelectedIndex;
    roboticArmTab.DataContext = selectedServoArray[selectedServoIndex];
    Console.WriteLine(selectedServoIndex);
}

private void VectorServo_Click(object sender, MouseButtonEventArgs e)
{
    switch ((sender as Rectangle).Name)
    {
        case "baseVector":
            if (selectedServoIndex != 0)
            {
                ServoComboBox.SelectedIndex = 0;
            }
            break;

        case "shoulderVector":
            if (selectedServoIndex != 1)
            {
                ServoComboBox.SelectedIndex = 1;
            }
            break;

        case "elbowVector":
            if (selectedServoIndex != 2)
            {
                ServoComboBox.SelectedIndex = 2;
            }
            break;

        case "armVector":
            if (selectedServoIndex != 3)
            {
                ServoComboBox.SelectedIndex = 3;
            }
            break;

        case "fingerVector":
            if (selectedServoIndex != 4)
            {
                ServoComboBox.SelectedIndex = 4;
            }
            break;
    }
}

```

```

        default:
            break;
    }
}

// Invoked when the engaged check box is clicked
private void EngagedCheckBox_Checked(object sender, RoutedEventArgs e)
{
    if (advServo.Attached)
    {
        selectedServoArray[selectedServoIndex].Engaged = true;
        advServo.servos[selectedServoIndex].Engaged = true;
    }
}

private void EngagedCheckBox_Unchecked(object sender, RoutedEventArgs e)
{
    if (advServo.Attached)
    {
        selectedServoArray[selectedServoIndex].Engaged = false;
        advServo.servos[selectedServoIndex].Engaged = false;
    }
}

private void ProgramWindow_MouseWheel(object sender, MouseWheelEventArgs e)
{
    if (e.Delta == 120)
    {
        PositionSlider.Value += 1;
    }
    else if (e.Delta == -120)
    {
        PositionSlider.Value -= 1;
    }
    Console.WriteLine(e.Delta);
}

private void ProgramWindow_Closing(object sender,
System.ComponentModel.CancelEventArgs e)
{
    Console.WriteLine("Closing");

    if (advServo != null)
    {
        if (advServo.Attached)

```

```

    {
        //disengage the servos
        for (int i = 0; i < NUMBER_OF_SERVOS; i++)
        {
            advServo.servos[i].Engaged = false;
        }
    }

    advServo.PositionChange -= new Phidgets.Events.PositionChangeEventHandler
        (advServo_PositionChange);
    advServo.VelocityChange -= new Phidgets.Events.VelocityChangeEventHandler
        (advServo_VelocityChange);

    advServo.close();
    advServo = null;
}
}

#endregion

#region Menu Items Event Handlers

private void AboutMenuItem_Click(object sender, RoutedEventArgs e)
{
    aboutWindow = new AboutWindow();
    aboutWindow.Show();
}

#endregion
}
}

```

#### Servo Class:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ComponentModel;

namespace Robotic_Arm_Control_Panel
{
    // This class implements INotifyPropertyChanged
    // to support one-way and two-way bindings

```

```

// (such that the UI element updates when the source
// has been changed dynamically)
class Servo : INotifyPropertyChanged
{
    // Declare the event
    public event PropertyChangedEventHandler PropertyChanged;

    private double position;
    private bool engaged;

    public double Position
    {
        get
        {
            return position;
        }

        set
        {
            if (Position != value)
            {
                position = value;
                // Call OnPropertyChanged whenever the property is updated
                OnPropertyChanged("Position");
            }
        }
    }

    public double Velocity { get; set; }
    public double Acceleration { get; set; }
    public double MaxPosition { get; set; }
    public double MinPosition { get; set; }
    public bool Engaged
    {
        get { return engaged; }
        set
        {
            if (Engaged != value)
            {
                engaged = value;
                OnPropertyChanged("Engaged");
            }
        }
    }
}

public Servo()

```

```

{
    //default values of each servo
    Position = 0;
    Velocity = 316;
    Acceleration = 182857;
    MaxPosition = 180;
    MinPosition = 0;
    Engaged = false;
}

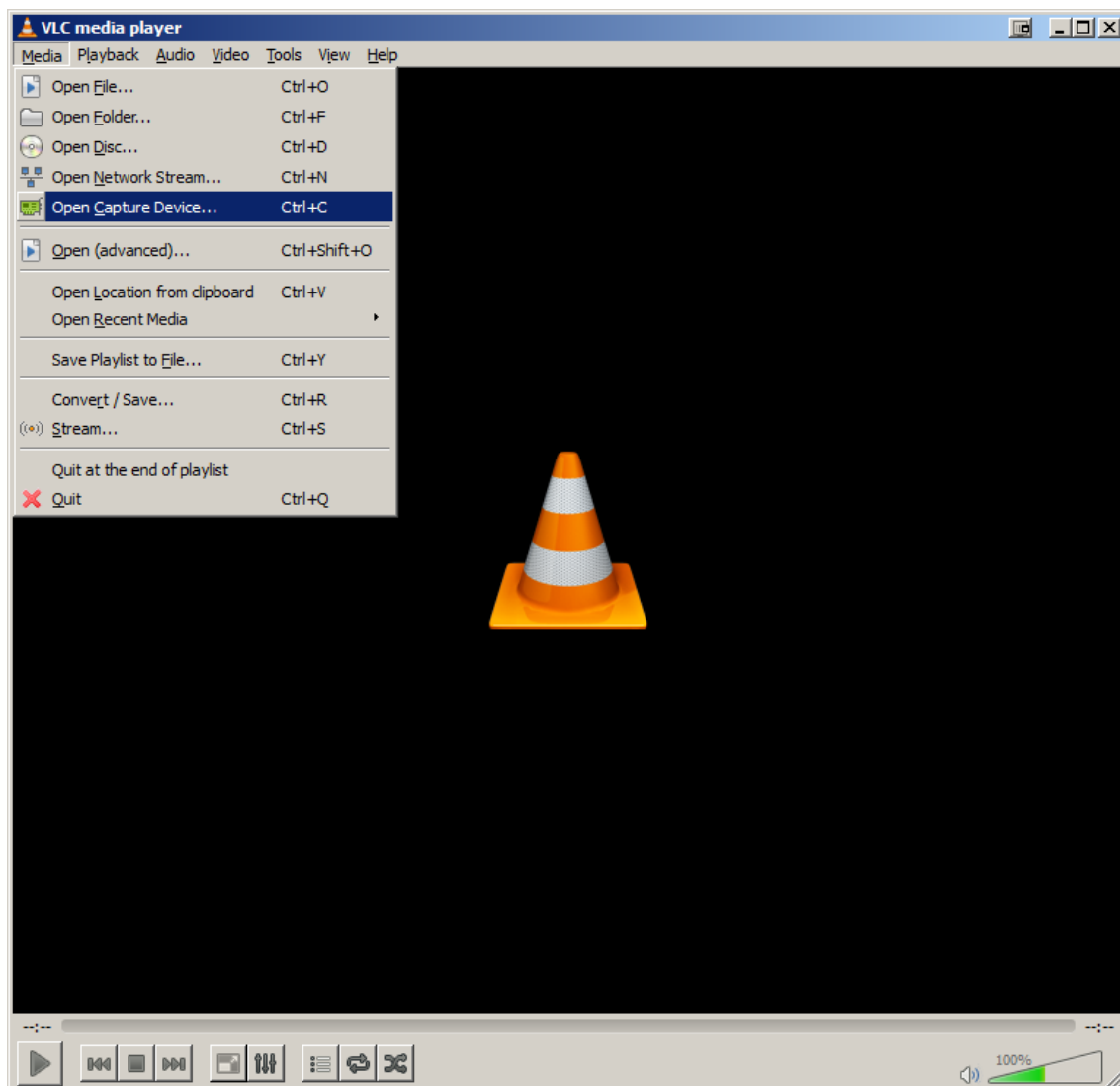
// Create the OnPropertyChanged method to raise the event
protected void OnPropertyChanged(string name)
{
    PropertyChangedEventHandler handler = PropertyChanged;
    if (handler != null)
    {
        handler(this, new PropertyChangedEventArgs(name));
    }
}
}

```

## **Appendix B. Broadcast Web Camera with VLC Media Player**

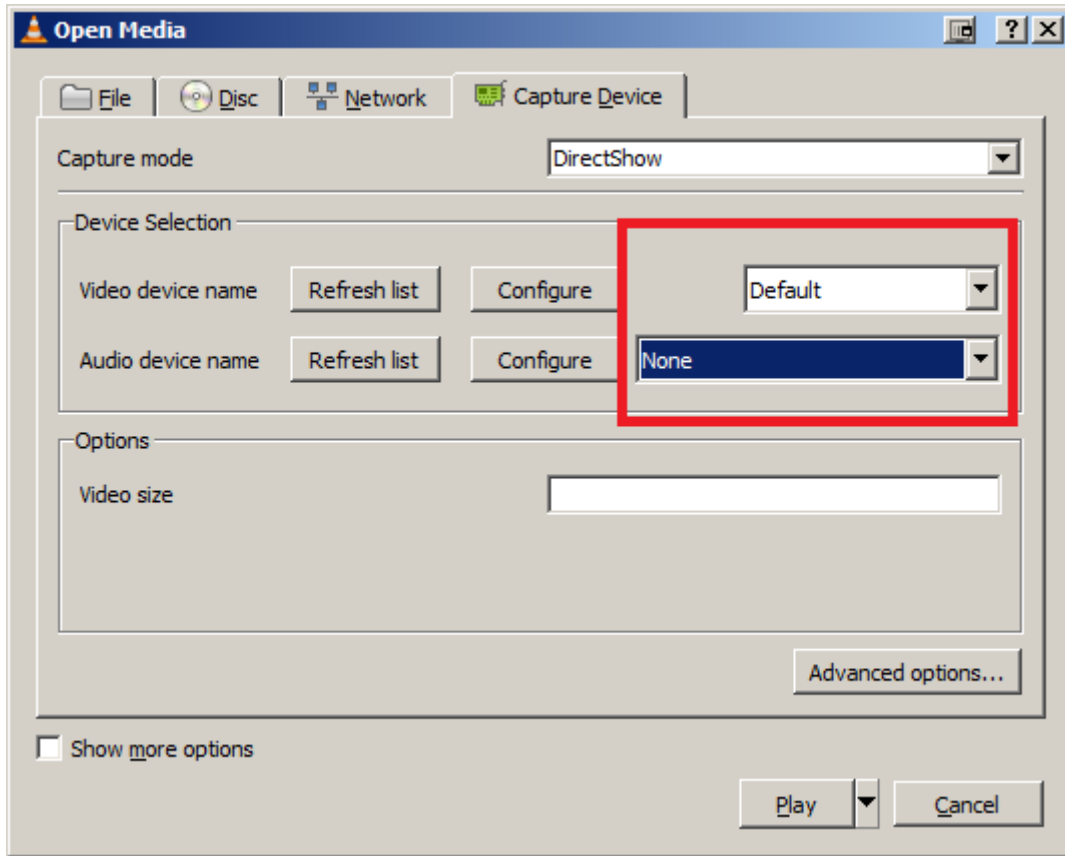
This section describes a step by step guide on how to stream the robotic arm's web cam using VLC media player.

1. Download and install the latest version of VLC media player from <http://www.videolan.org/vlc/index.html>
2. Open VLC media player, click > Media > Open Capture Device (Figure 22)



**Figure 22**

3. Select “Microsoft LifeCam” in the “Video device name” section (Figure 23).
4. Select “none” in the “Audio device name” section (Figure 23).



**Figure 23**

5. Click the pick button located to the right of the play button, and select “stream” (Figure 24).

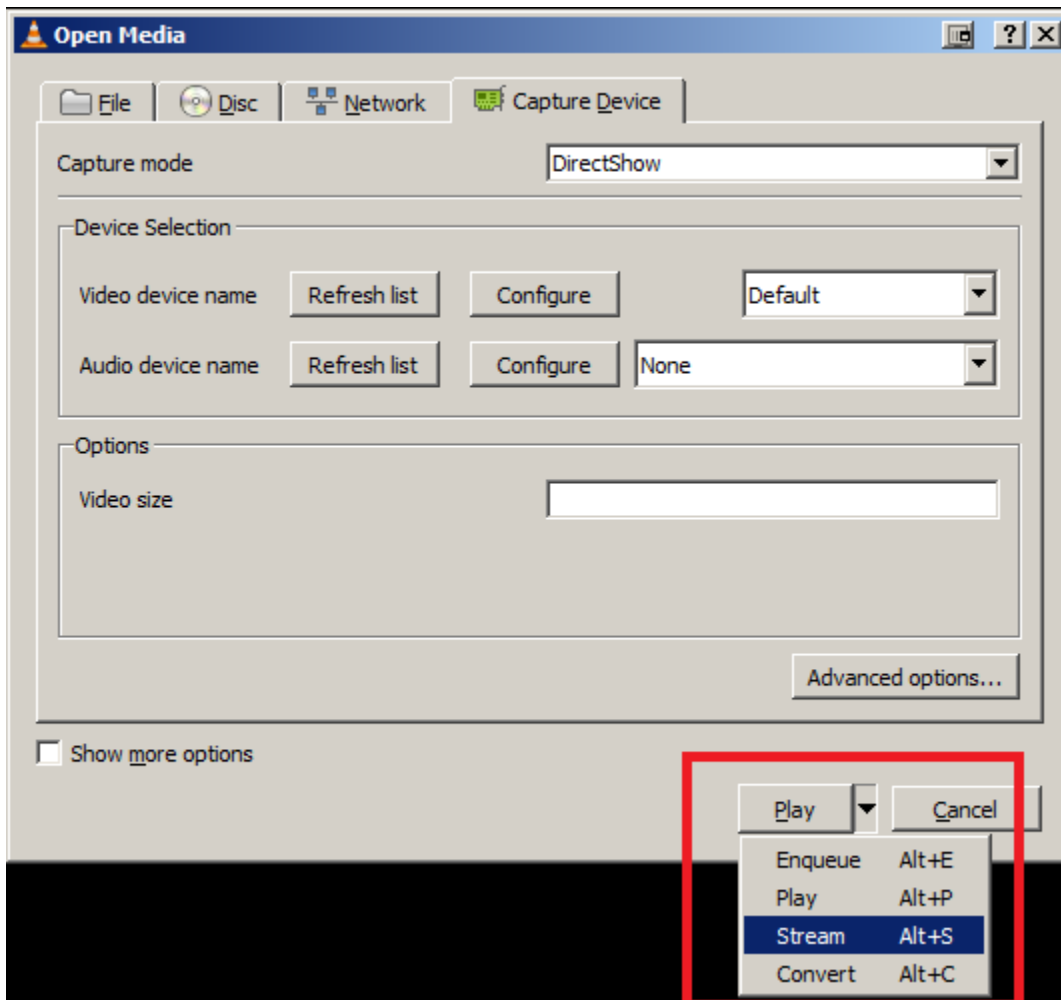
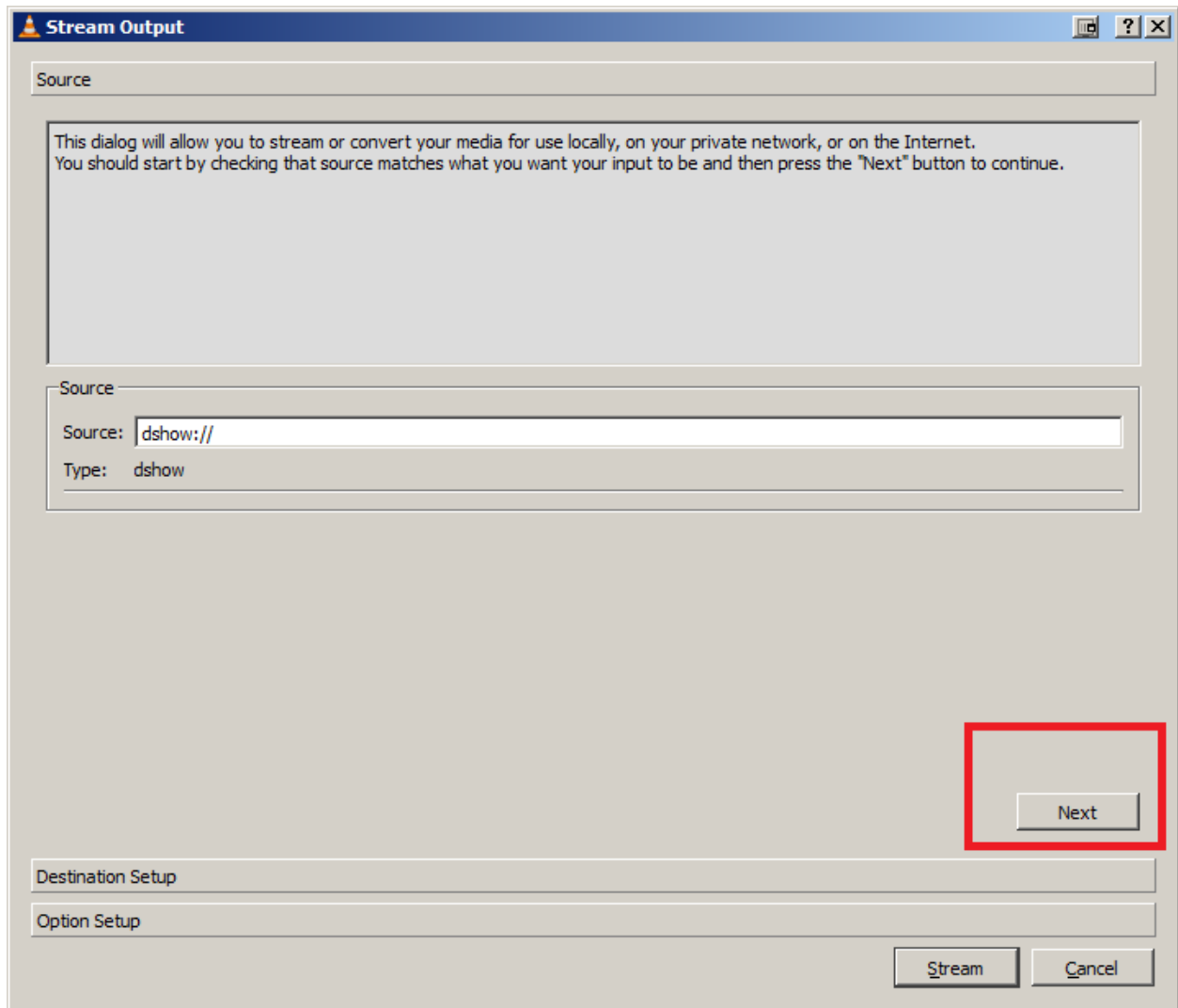


Figure 24

6. The “Stream Output” window will open, click “Next” (Figure 25).



**Figure 25**

7. Select “MS-WMSP” in the “New destination” section as the streaming method. Click the “Add” button (Figure 26).

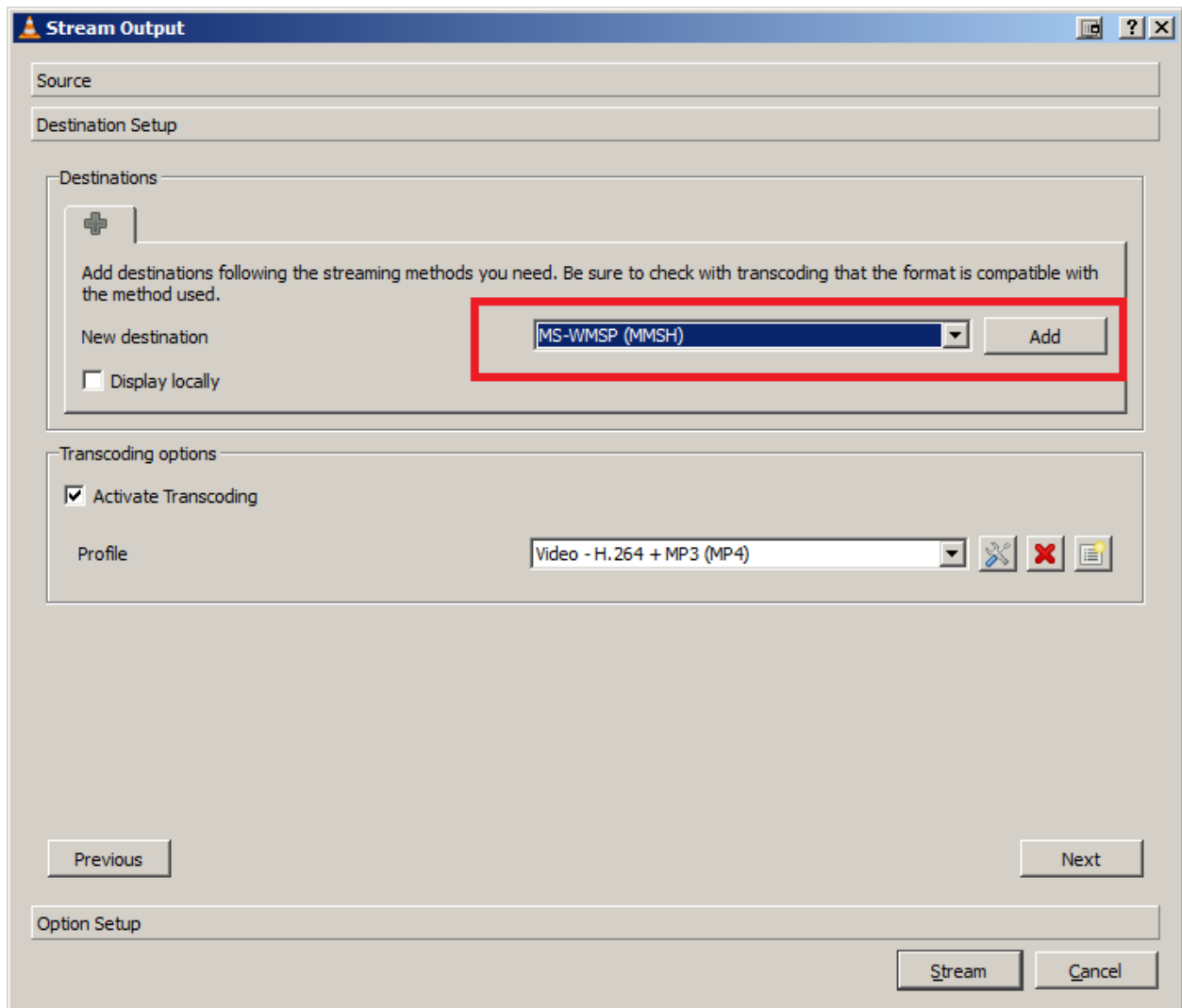


Figure 26

8. Enter CoroBot's public IP address in the "Address" input box. Enter 4668 in the "Port" input box (Figure 27).

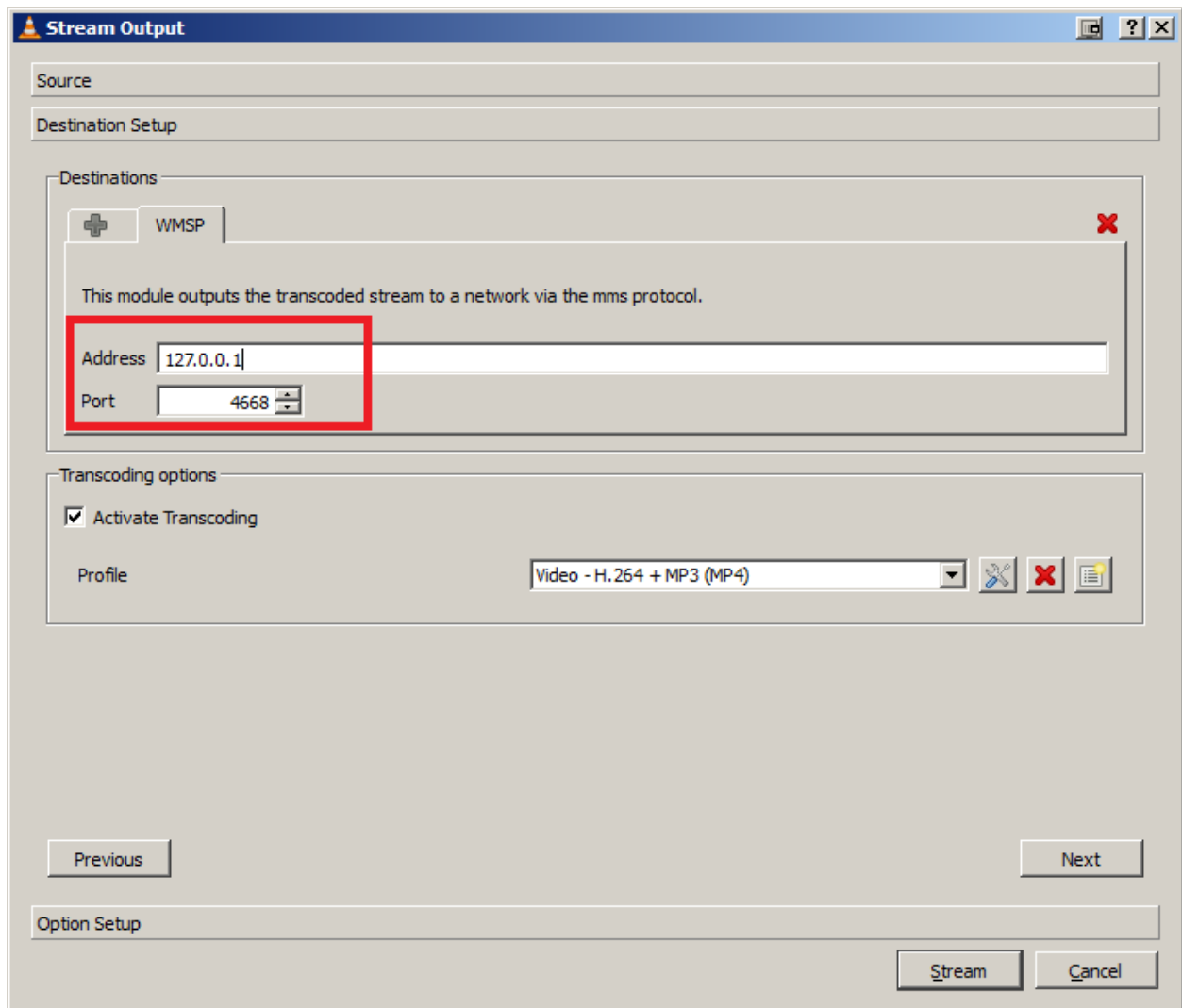


Figure 27

9. Select “Video-WMV + WMA (ASF)” as the transcoding option, and click “Next” (Figure 28).

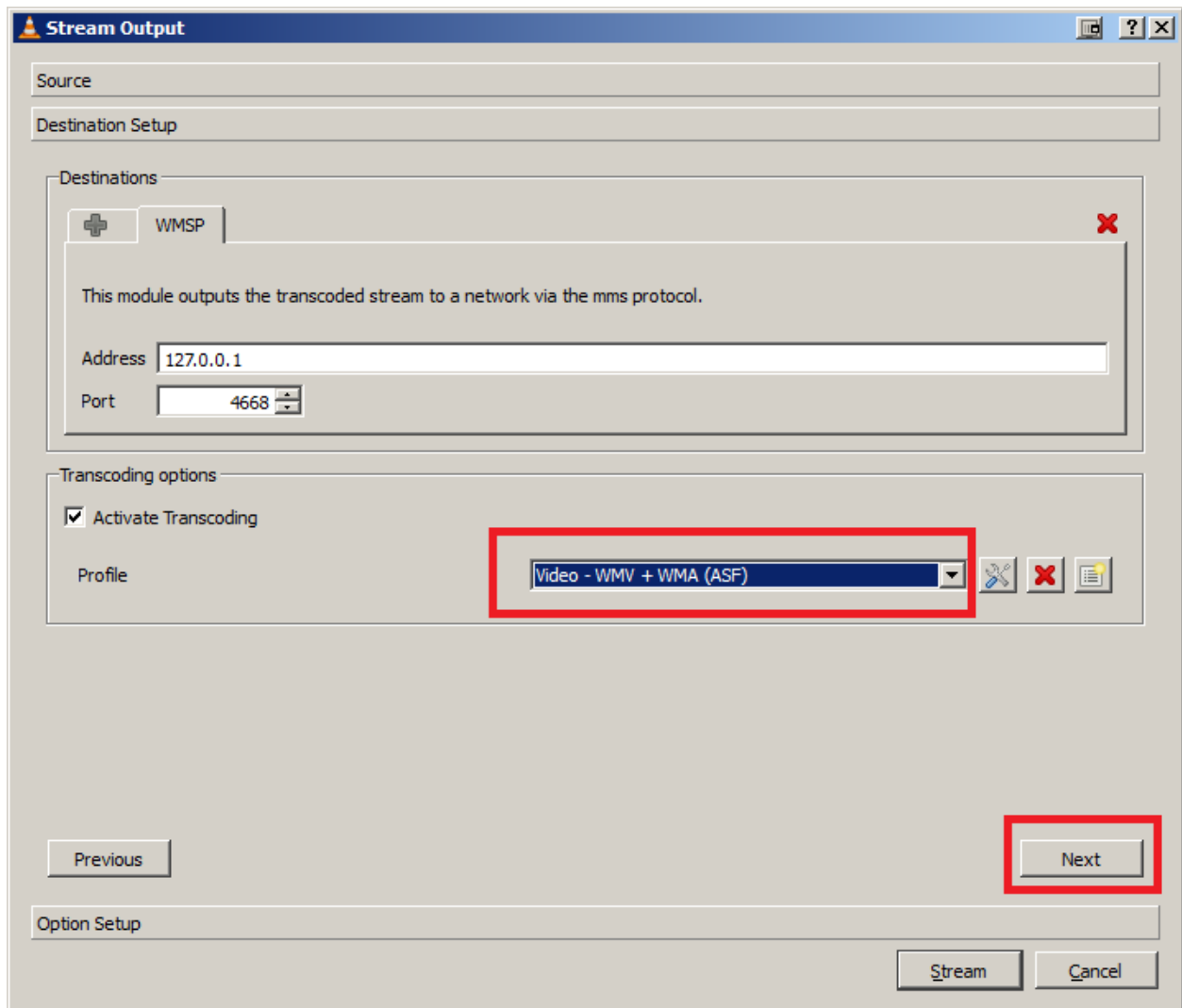
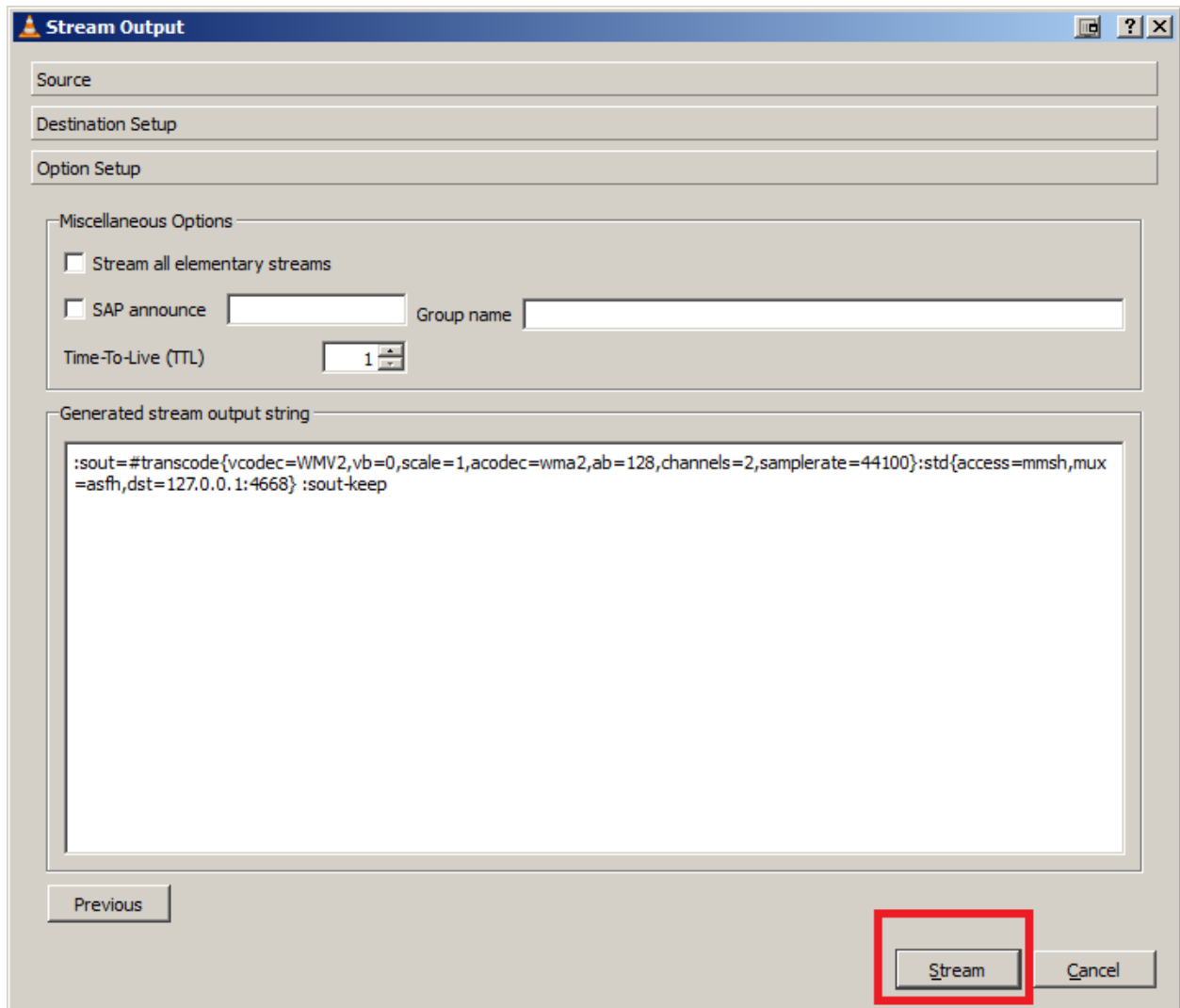


Figure 28

10. Finally, click the “Stream” button to stream the web cam over the network (Figure 29).



**Figure 29**

## **Appendix C. Build Instructions**

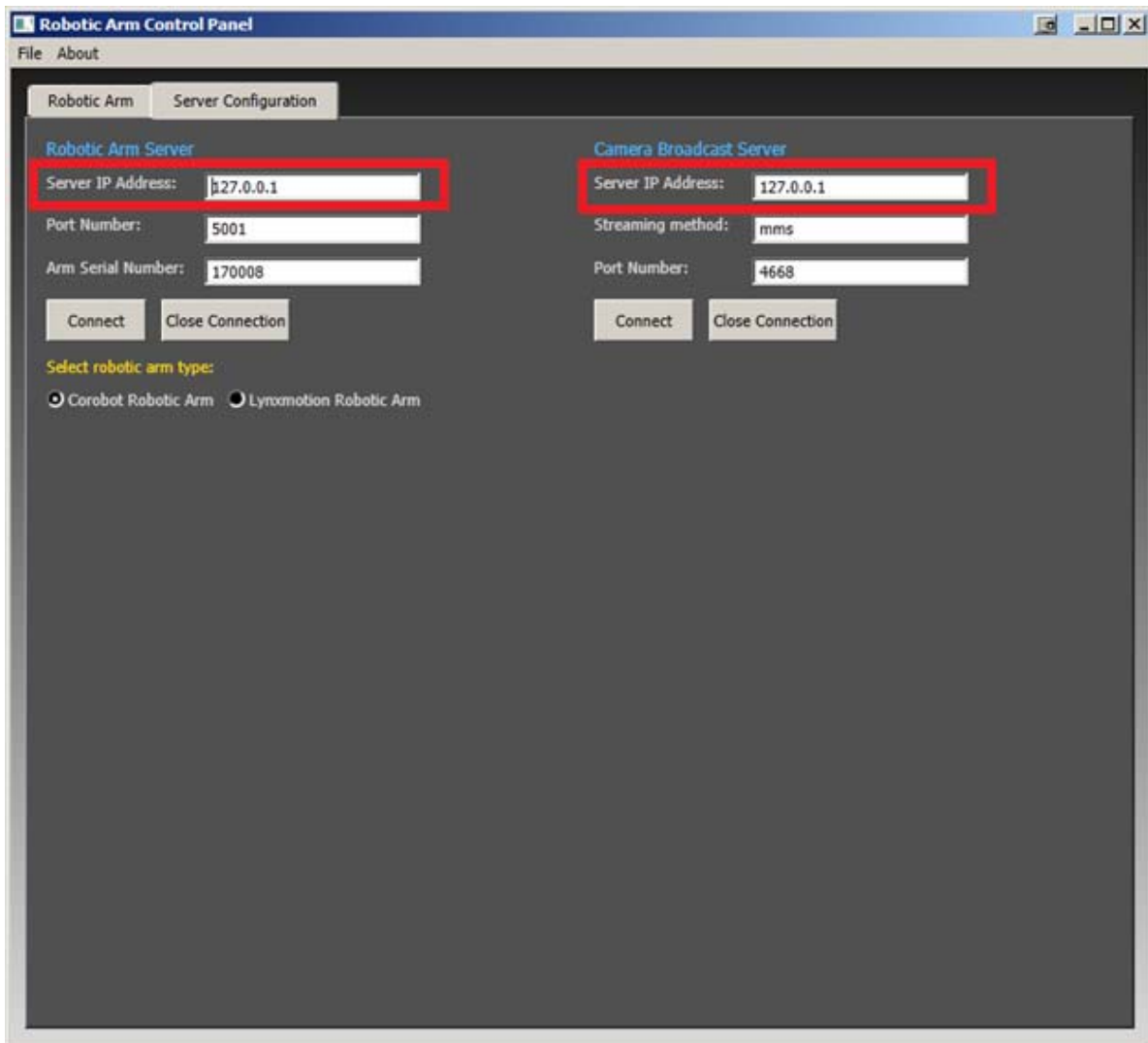
This section describes a step by step guide on how to build and run this project.

1. Download Microsoft .NET Framework 3.5 or higher  
<http://www.microsoft.com/en-us/download/details.aspx?id=21>
2. Download and install Microsoft Visual Studio 2010  
<http://www.microsoft.com/visualstudio/eng/downloads#d-2010-express>
3. Download the Phidget installer which installs the Phidget drivers and libraries for Windows  
[http://www.phidgets.com/docs/OS\\_-\\_Windows#Quick\\_Downloads](http://www.phidgets.com/docs/OS_-_Windows#Quick_Downloads)
4. Open Visual Studio 2010
5. Click File > Open > Project/Solution > Select the Robotic Arm Control Panel (.sln) file.
6. Click Build > Build Solution (F6)
7. Click Build > Build Robotic Arm Control Panel (Shift + F6)
8. To run the program click Debug > Start Debugging (F5)

## **Appendix C. User Manual**

This section demonstrates how to use the Robotic Arm Control Panel application to control the robotic arm.

1. Open the Robotic Arm Control Panel application and click the “Server Configuration” tab (Figure 30).
2. Enter CoroBot’s public IP address in the “Server IP Address” input box of the “Robotic Arm Server” section and the “Camera Broadcast Server” section. Leave the rest of the input boxes as they appear in Figure 30. Click the “Connect” button in each section.



**Figure 30**

3. Click the “Robotic Arm” tab and select a servo by either left-clicking it in the vector graphic or in the “Select Servo” combo box (Figure 31).

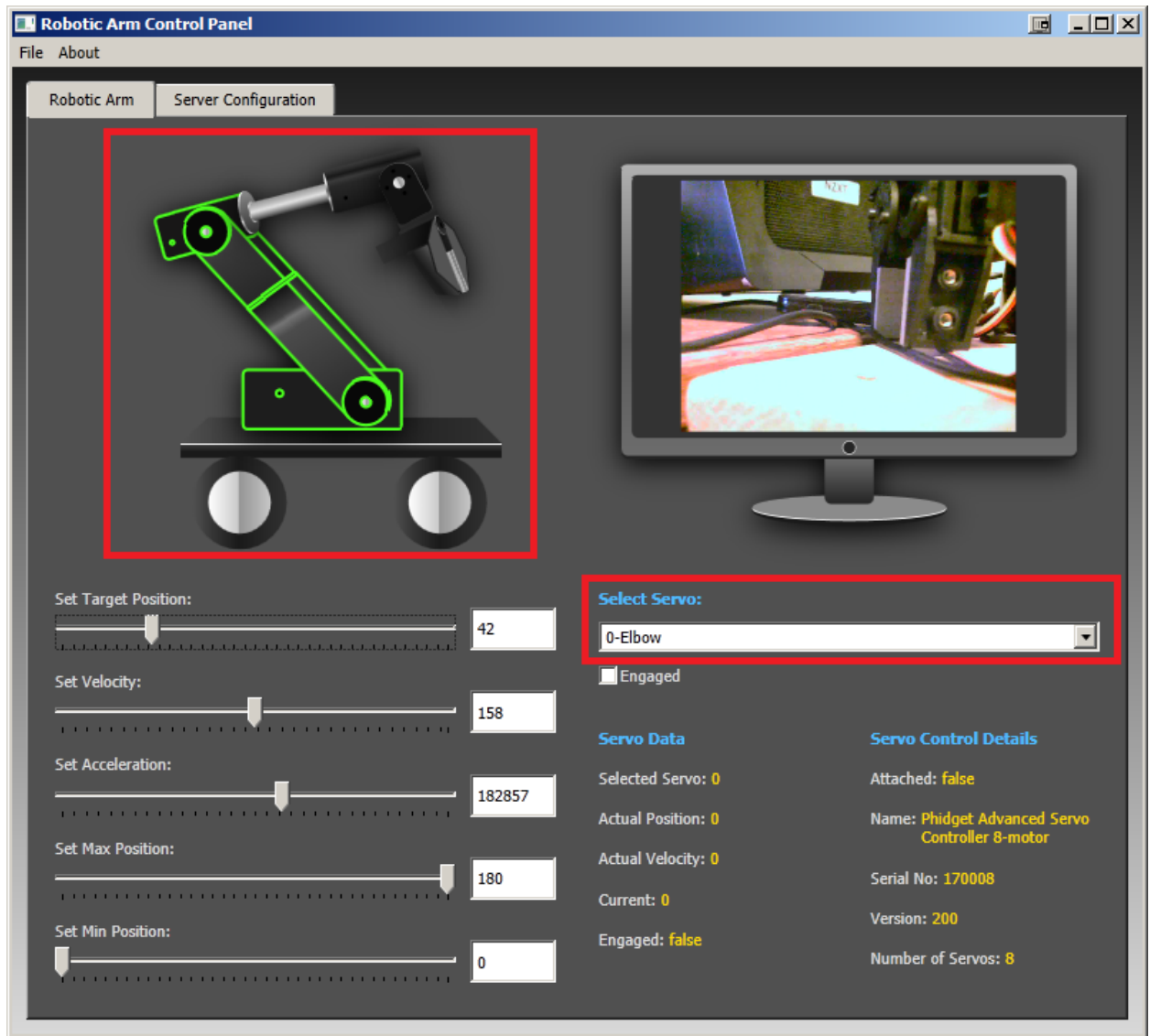


Figure 31

4. Change the property of the selected servo, such as its position, by moving the corresponding slider (Figure 32).

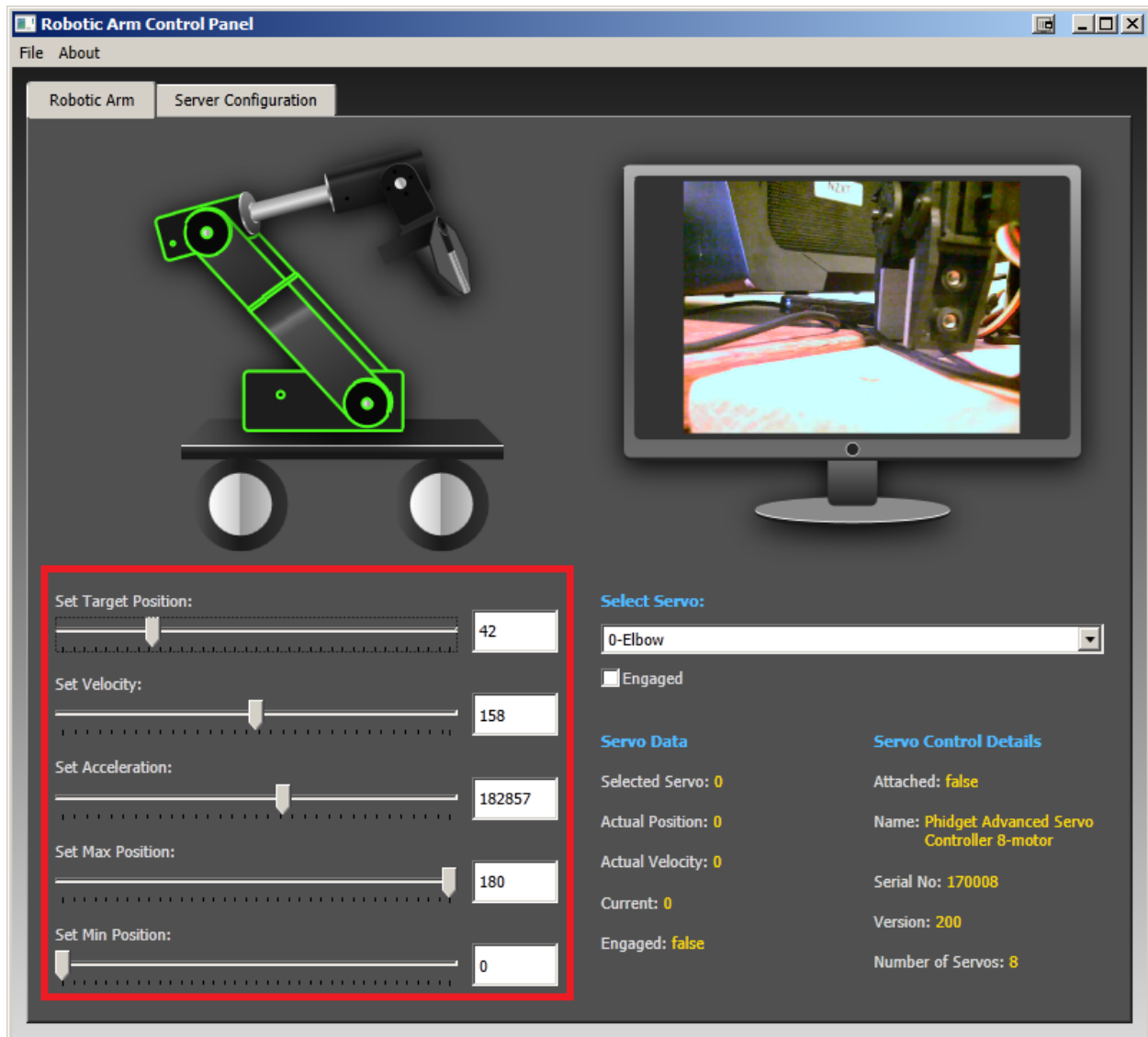


Figure 32